

```

DDDDDDDDDDDDDD RRRRRRRRRRRR IIIIIIIIII VVV VVV EEEEEEEEEEEEEEE RRRRRRRRRRRR
DDDDDDDDDDDDDD RRRRRRRRRRRR IIIIIIIIII VVV VVV EEEEEEEEEEEEEEE RRRRRRRRRRRR
DDDDDDDDDDDDDD RRRRRRRRRRRR IIIIIIIIII VVV VVV EEEEEEEEEEEEEEE RRRRRRRRRRRR
DDD DDD DDD RRR RRR RRR VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR RRR VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR RRR VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR RRR VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR RRR VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR RRR VVV VVV EEE RRR RRR
DDD DDD DDD RRRRRRRRRRRR IIIIIIIIII VVV VVV EEEEEEEEEEEEE RRRRRRRRRRRR
DDD DDD DDD RRRRRRRRRRRR IIIIIIIIII VVV VVV EEEEEEEEEEEEE RRRRRRRRRRRR
DDD DDD DDD RRRRRRRRRRRR IIIIIIIIII VVV VVV EEEEEEEEEEEEE RRRRRRRRRRRR
DDD DDD DDD RRR RRR IIIIIIIIII VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR IIIIIIIIII VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR IIIIIIIIII VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR RRR IIIIIIIIII VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR RRR IIIIIIIIII VVV VVV EEE RRR RRR
DDD DDD DDD RRR RRR RRR IIIIIIIIII VVV VVV EEE RRR RRR
DDDDDDDDDDDDDD RRR RRR RRR IIIIIIIIII VVV VVV EEEEEEEEEEEEE RRR RRR
DDDDDDDDDDDDDD RRR RRR RRR IIIIIIIIII VVV VVV EEEEEEEEEEEEE RRR RRR
DDDDDDDDDDDDDD RRR RRR RRR IIIIIIIIII VVV VVV EEEEEEEEEEEEE RRR RRR

```

[illegible]

```

NN      NN      000000      DDDDDDDD      RRRRRRRR      IIIIII      VV      VV      EEEEEEEEEEE      RRRRRRRR
NN      NN      000000      DDDDDDDD      RRRRRRRR      IIIIII      VV      VV      EEEEEEEEEEE      RRRRRRRR
NN      NN      00      00      DD      DD      RR      RR      RR      RR      EE      EE      RR      RR
NNNN      NN      00      00      DD      DD      RR      RR      RR      RR      EE      EE      RR      RR
NNNN      NN      00      00      DD      DD      RR      RR      RR      RR      EE      EE      RR      RR
NN      NN      00      00      DD      DD      RRRRRRRR      IIII      VV      VV      EEEEEEEEE      RRRRRRRR
NN      NN      00      00      DD      DD      RRRRRRRR      IIII      VV      VV      EEEEEEEEE      RRRRRRRR
NN      NNNN      00      00      DD      DD      RR      RR      RR      RR      EE      EE      RR      RR
NN      NNNN      00      00      DD      DD      RR      RR      RR      RR      EE      EE      RR      RR
NN      NN      00      00      DD      DD      RR      RR      RR      RR      EE      EE      RR      RR
NN      NN      00      00      DD      DD      RR      RR      RR      RR      EE      EE      RR      RR
NN      NN      000000      DDDDDDDD      RR      RR      IIIIII      VV      VV      EEEEEEEEEEE      RR      RR
NN      NN      000000      DDDDDDDD      RR      RR      IIIIII      VV      VV      EEEEEEEEEEE      RR      RR

```

```

LL      IIIIII      SSSSSSSS
LL      IIIIII      SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLL      IIIIII      SSSSSSSS

```


(7)	278	Standard Driver Tables
(14)	636	P2 buffer verification tables
(17)	783	NOSCONTROL_INIT - Initialize Sync line device
(18)	818	NOSUNIT_INIT - Initialize the device unit
(20)	877	NOSXMITFDT - Transmit I/O FDT routine
(21)	929	COM_XMITFDT - Common transmit FDT routine
(22)	1019	NOSALT_ENTRY - Alternate I/O entry
(23)	1103	NOSRCVFDT - Receive I/O FDT routine
(24)	1189	NOSSETMODEFDT, Set mode I/O operation FDT routine
(25)	1323	SETMODE_CTRL, Perform setmode FDT operation on controller
(26)	1483	ALLOC_BUFFER, Allocate a buffer
(27)	1514	INIT_NO_BUFFER - Initialize NOB extension
(28)	1591	NOSSENSEMODEFDT, Sense Mode I/O operation FDT routine
(29)	1715	SENSEMODE_CTRL, Perform SENSEMODE FDT processing for controller
(30)	1825	SENSE_MODEM - Perform SENSEMODE READ_MODEM FDT processing
(31)	1869	GET_CHAR_BUFS, Get P1 and P2 characteristics buffers
(32)	1937	CHECK_BUFS, Check P1 and P2 buffers for write access
(33)	1986	CHECK_P1, Check P1 buffer address for write access
(34)	2024	ALLOC_P2BUF, Allocate a P2 buffer and charge user's quota
(35)	2088	CHG_UCB_NOB, Change UCB and the NOB parameter values
(37)	2181	CHG_TRIB, Change trib parameter values
(38)	2230	NOSSTARTIO - Start setmode I/O operation
(39)	2286	START_TRANSMIT - Start transmit I/O
(40)	2376	GET_XMT - Get next transmit to send
(41)	2448	READ_MODEM - Read device modem register
(42)	2471	FILL_DUE_TIME_TABLE - Set up the due time table
(43)	2522	START - Start unit, device and/or protocol
(46)	2739	Set protocol characteristic
(47)	2792	FILL_FREELIST - FILL MESSAGE FREE LIST
(48)	2854	START_RECEIVE and GET_RECEIVE - Start receive and Get a receive buff
(49)	2937	NOSGETNXT - Class driver GETNXT routine
(50)	3186	NOSPUTNXT - Async class driver receive character routine
(51)	3384	NOSCLASS_PORTFORK - Async DDCMP fork routines
(52)	3447	NOSSETUP_UCB - Async DDCMP set up UCB
(53)	3466	NOSPORT_TRANSITION - Async DDCMP port transition routine
(55)	3493	NOSREADERROR - Async DDCMP read error routine
(57)	3573	NOSCLASS_DISCONNECT - Async DDCMP class disconnect
(58)	3593	NOSCLASS_POWERACTION - Async DDCMP power action
(59)	3612	NOSNULL - Async DDCMP null routine
(61)	3632	FORKDONE - Fork process
(62)	3708	RECEIVE_DONE - Complete a receive buffer
(63)	3800	FINISH_RCV_IO - Finish receive I/O processing
(65)	3869	TRANSMIT_DONE - Transmit completion routine
(68)	4011	NOSREGDUMP - Error log and diagnostics register dump
(69)	4033	Poke user process on attention condition
(70)	4068	TIMEOUT - TIMEOUT
(71)	4203	NOSCANCEL - Cancel I/O routine
(72)	4249	CLEAR_NO_BUFFER - Clear NO buffer and disconnect the port
(74)	4292	SHUTDOWN - Shut down unit, device and/or protocol
(77)	4533	VALIDATE_P2, Validate P2 buffer parameters
(77)	4534	VALIDATE_P2_TRIB, Validate P2 buffer with Trib param
(77)	4535	VALIDATE_P2_UCB, Validate P2 buffer with UCB
(78)	4662	UPDATE_P2, Update UCB/TRIB based on P2 buffer parameters
(79)	4737	RETURN_P2, Return UCB/DDCMP buffer parameters
(80)	4797	UNPACK_P2_BUF, Unpack a P2 parameter from P2 buffer

```

0000 1      .TITLE  NODRIVER- VAX/VMS DMF32 Async DDMCP Line Driver
0000 2      .IDENT  'V04-001'
0000 3
0000 4
0000 5      *****
0000 6      *
0000 7      *  COPYRIGHT (c) 1978, 1980, 1982, 1984
0000 8      *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9      *  ALL RIGHTS RESERVED.
0000 10     *
0000 11     *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12     *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13     *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14     *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15     *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16     *  TRANSFERRED.
0000 17     *
0000 18     *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19     *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20     *  CORPORATION.
0000 21     *
0000 22     *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23     *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24     *
0000 25     *
0000 26     *****
0000 27     ++
0000 28     FACILITY:
0000 29
0000 30     VAX/VMS Async DDCMP line driver
0000 31
0000 32     ABSTRACT:
0000 33
0000 34     This module contains the Async DDCMP line driver FDT routines,
0000 35     interrupt dispatcher, interrupt service and fork routines.
0000 36
0000 37     AUTHOR:
0000 38
0000 39     Meg Dumont 1-OCT-83
0000 40
0000 41     MODIFIED BY:
0000 42
0000 43     V04-001 MMD0328      Meg Dumont,      5-Sep-1984  18:16
0000 44     Fix bugs in the xmt error, error paths.
0000 45
0000 46     V03-008 MMD0322      Meg Dumont,      13-Aug-1984  8:55
0000 47     Fix to SENSEMODE_CTRL where DDCMP could be called without
0000 48     the protocol/NOB buffers having been set up.
0000 49
0000 50     V03-007 MMD0318      Meg Dumont,      25-Jul-1984  11:34
0000 51     Fix to clear the GFB before giving it to protocol and fix
0000 52     to DEVTIMER, we were using the wrong register in testing the
0000 53     timeout conditions.
0000 54
0000 55     V03-006 LMP0275      L. Mark Pilant,    12-Jul-1984  12:30
0000 56     Initialize the ACL info in the ORB to be a null descriptor
0000 57     list rather than an empty queue. This avoids the overhead

```



```

0000 58 : of locking and unlocking the ACL mutex, only to find out
0000 59 : that the ACL was empty.
0000 60 :
0000 61 : V03-005 MMD0299 Meg Dumont, 18-May-1984 10:26
0000 62 : Fix to returning of quota on line startup. Also fix to check
0000 63 : that the buffers are allocated before using them in the
0000 64 : sensemode paths'.
0000 65 :
0000 66 : V03-004 MMD0297 Meg Dumont, 14-May-1984 14:54
0000 67 : Fix bug in COM_XMITFDT error path
0000 68 :
0000 69 : V03-003 MMD0294 Meg Dumont, 23-Apr-1984 16:02
0000 70 : Fix so that if the TFB address is not valid the driver won't
0000 71 : try to do an XMIT. Fix to support a new interface with the
0000 72 : protocol. This interface is designed to fix problems with
0000 73 : retransmitting messages out of order.
0000 74 :
0000 75 : V03-002 LMP0221 L. Mark Pilant, 27-Mar-1984 10:13
0000 76 : Change UCBSL_OWNUIC to ORBSL_OWNER and UCBSW_VPROT to
0000 77 : ORBSW_PROT.
0000 78 :
0000 79 : V03-001 MMD0262 Meg Dumont, 22-Mar-1984 14:40
0000 80 : Fix so that the driver doesn't crash if an XMT or RCV
0000 81 : is queued to the device before the device is started.
0000 82 : Add support for timers on receives and transmits.
0000 83 :
0000 84 :--

```



```
0000 86
0000 87 ; System definitions
0000 88
0000 89 $ACBDEF
0000 90 $CRBDEF
0000 91 $CXBDEF
0000 92 $DDBDEF
0000 93 $DDCMPDEF
0000 94 $DEVDEF
0000 95 $DLADEF
0000 96 $DLKDEF
0000 97 $DYNDEF
0000 98 $DPTDEF
0000 99 $GFDEF
0000 100 $IODEF
0000 101 $IPLDEF
0000 102 $IRPDEF
0000 103 $JIBDEF
0000 104 $MFDDEF
0000 105 $NMADEF
0000 106 $ORBDEF
0000 107 $PCBDEF
0000 108 $SSDEF
0000 109 $TFDEF
0000 110 $TTDEF
0000 111 $TT2DEF
0000 112 $TTYDEF
0000 113 $TTYMACS
0000 114 $TTYDEFS
0000 115 $TTYMODEM
0000 116 $TTYUCBDEF
0000 117 $TTYVECDDEF
0000 118 $TQDEF
0000 119 $UCBDEF
0000 120 $VECDDEF
0000 121 $XMDEF
0000 122 $XMTQDEF
0000 123
0000 124
0000 125 ; Local macros
0000 126
0000 127 .MACRO SETBIT POS,BAS,?L
0000 128 BBSS POS,BAS,L
0000 129 L:
0000 130 .ENDM SETBIT
0000 131
0000 132 .MACRO CLRBIT POS,BAS,?L
0000 133 BBCC POS,BAS,L
0000 134 L:
0000 135 .ENDM CLRBIT
0000 136
0000 137 ; Bit Definitions
0000 138
0000 139 .MACRO BITDEF BLK,SYM,BITVAL
0000 140
0000 141 'BLK' V 'SYM' = BITVAL
0000 142 'BLK' M 'SYM' = 1<BITVAL>
```

```
: AST control
: Controller request block
: Complex buffer block
: Device data block
: Constant def's for DDCMP
: Device characteristics
: Driver/protocol command defs
: Driver/protocol command defs
: Dynamic data structures
: Driver prologue table
: Global field definitions
: I/O function codes
: Interrupt priority levels
: I/O packets
: Job information block
: Message field descriptor
: Network management def's
: Object's Rights Block offsets
: Process control block
: System service status
: DDCMP buffer def's
: Terminal characteristics
: " "
: Def terminal macros
: Define terminal definitions
: Define terminal modem signals
: Terminal UCB extension
: Define terminal vectors
: Timer queue element def's
: Unit control block
: XMDRIVER symbols
: Transmit Q def's
```

```
: Set a single bit
```

```
: Clear a single bit
```

```

0000 143
0000 144      .ENDM
0000 145
0000 146 ; Constant definitions
0000 147
0000 148      .MACRO  CONSTDEF BLK,SYM,CONSTVAL
0000 149
0000 150      'BLK'$C_'SYM' = CONSTVAL
0000 151      .ENDM
0000 152
0000 153 ; Set up the duetim table
0000 154
0000 155      .MACRO  STORE_DUETIM BAUDRATE,BITS,RESULT1,RESULT2,TABLE
0000 156
0000 157          DIVW3    #'BAUDRATE',BITS,RESULT1
0000 158          ADDW2    #1,RESULT1
0000 159          MOVZBL   #TISC_BAUD 'BAUDRATE',RESULT2
0000 160          MOVW     RESULT1,TABLE[RESULT2]
0000 161
0000 162      .ENDM  STORE_DUETIM

```



```
0000 164 .MACRO PARAM TYPE,OFFSET,WIDTH,MIN,MAX,INVALID,BASE
0000 165 :++
0000 166 :
0000 167 :
0000 168 :
0000 169 :
0000 170 :
0000 171 :
0000 172 :
0000 173 :
0000 174 :
0000 175 :--
0000 176 .IF BLANK TYPE
0000 177 .WORD 0
0000 178 .IF FALSE
0000 179 $$$TYP = TYPE & PRM_M_TYPE ; Isolate type code
0000 180 .IIF NOT_BLANK <MIN>, $$$TYP = $$$TYP!PRM_M_MIN
0000 181 .IIF NOT_BLANK <MAX>, $$$TYP = $$$TYP!PRM_M_MAX
0000 182 .IIF NOT_BLANK <INVALID>, $$$TYP = $$$TYP!PRM_M_INVALID
0000 183 .WORD $$$TYP
0000 184 $$$OFF = OFFSET & OFF_M_VALUE ; Isolate offset only
0000 185 $$$WID = 0 ; Set null width
0000 186 .IIF IDN <WIDTH><B>, $$$WID = <1@OFF_V_WIDTH>
0000 187 .IIF IDN <WIDTH><W>, $$$WID = <2@OFF_V_WIDTH>
0000 188 .IIF IDN <WIDTH><L>, $$$WID = <3@OFF_V_WIDTH>
0000 189 .WORD $$$OFF!$$$WID
0000 190 .IIF NOT_BLANK <MIN>, .WORD MIN
0000 191 .IIF NOT_BLANK <MAX>, .WORD MAX
0000 192 .IIF NOT_BLANK <INVALID>, .WORD INVALID
0000 193 'BASE'_PRM_BUFSIZ = 'BASE'_PRM_BUFSIZ + 6
0000 194 .ENDC
0000 195 .ENDM PARAM
0000 196
0000 197 .MACRO SKIP BIT,LOC,REG,CONTEXT=W,?L ; SKIP FIELD
0000 198 BBC #BIT,LOC,L ; Br if field not present
0000 199 TST'CONTEXT (REG)+ ; Skip next field
0000 200 L:
0000 201 .ENDM SKIP
```



```
0000 203
0000 204 :++
0000 205 : This macro translates into the CASEx instruction. It calculates the
0000 206 : "base" and "limit" parameters from the <index,displacement> list
0000 207 : specified in the 'vector' parameter. The dispatch table is set up
0000 208 : such that any unspecified index value within the bounds of the
0000 209 : transfer vector is associated with a displacement which transfers
0000 210 : control to the first location after the CASE statement, i.e., behaves
0000 211 : as if the index were out of bounds.
0000 212
0000 213 : Example:
0000 214 : $DISPATCH      R0,<-                ; Message type in R0
0000 215 :                ;index displacement
0000 216 :
0000 217 :                <CI,   NSP$RCV_CI>,-    ; Process CI message
0000 218 :                <CC,   NSP$RCV_CC>,-    ; Process CC message
0000 219 :                <DI,   NSP$RCV_DI>,-    ; Process DI message
0000 220 :                <DC,   NSP$RCV_DI>,-    ; Process DC message
0000 221 :
0000 222 :                >
0000 223 : BRW      NSP$RCV_ILLMSG                ; Message type unknown
0000 224 : --
0000 225 : MACRO $DISPATCH,      INDX,VECTOR,TYPE=W,NMODE=S^#,?MN,?MX,?S,?SS,?ZZ
0000 226 : SS:
0000 227
0000 228 : .MACRO $DSP1,$DSP1 1
0000 229 : .IRP   $DSP1_2,$DSP1_1
0000 230 :       $DSP2-$DSP1_2
0000 231 : .ENDR
0000 232 : .ENDM
0000 233
0000 234 : .MACRO $DSP2,$DSP2 1,$DSP2 2
0000 235 : .=<$DSP2 1-MN>*2 + 5
0000 236 : .WORD  $DSP2_2-S
0000 237 : .ENDM
0000 238
0000 239
0000 240 : .MACRO $BND1,$BND1 1,$BND1 2,$BND1 3
0000 241 : $BND2  $BND1_1,$BND1_2
0000 242 : .ENDM
0000 243
0000 244 : .MACRO $BND2,$BND2 1,$BND2 2
0000 245 : .IIF  $BND2_1,$BND2_2-.,      .=$BND2_2
0000 246 : .ENDM
0000 247
0000 248 : .MACRO $BND  $BND 1,$BND 2
0000 249 : .IRP   $BND_3,<$BND_2>
0000 250 :       $BNDT  $BND_1,$BND_3
0000 251 : .ENDR
0000 252 : .ENDM
0000 253
0000 254 : .=0
0000 255 : ZZ:
0000 256 : $BND  GT,<VECTOR>
0000 257 : MX:
0000 258 : $BND  LT,<VECTOR>
0000 259 : MN:
```



```

0000 260      . = SS
0000 261
0000 262 CASE 'TYPE      INDX, # < MN - ZZ >, NMODE ' < MX - MN >
0000 263 S:
0000 264      . REPT      MX - MN + 1
0000 265      . WORD      < MX - MN > * 2 + 2
0000 266      . ENDR
0000 267
0000 268      . = S
0000 269
0000 270      $DSP1      << VECTOR >>
0000 271
0000 272      . = < MX - MN > * 2 + S + 2
0000 273
0000 274      . ENDM
0000 275
0000 276

```

```

0000 278 .SBTTL Standard Driver Tables
0000 279
00000000 280 .PSECT $$$105_PROLOGUE
0000 281
0000 282 ; Driver Prologue Table
0000 283
0000 284 NOSDPT:
0000 285
0000 286 DPTAB -
0000 287 END=DRIVER END,- ; End of driver plus DDCMP
0000 288 FLAGS=DPTSM NOUNLOAD,- ; Set driver can't be reloaded
0000 289 ADAPTER=NULL,- ; no hardware assoc with driver
0000 290 UCBSIZE=UCBS%_TL_LENGTH,- ; UCB size
0000 291 NAME=NODRIVER,- ; Driver name
0000 292 VECTOR=ADDCMP_VECTOR ; Async DDCMP vector table
0038 293
0038 294 DPT_STORE INIT ; Initialization data
0038 295 DPT_STORE UCB,UCBS%_FIPL,B,8 ; Fork IPL
003C 296 DPT_STORE UCB,UCBS%_DIPL,B,21 ; Device IPL
0040 297 DPT_STORE UCB,UCBS%_DEVCHAR,L,<- ; Device characteristics
0040 298 DEVSM_NET!DEVSM_IDV!DEVSM_ODV>
0047 299 DPT_STORE UCB,UCBS%_DEVCLASS,B,DC%_SCOM ; Device class
004B 300 DPT_STORE UCB,UCBS%_DEVTYPE,B,DTS_ADDCMP ; Device type
004F 301 DPT_STORE UCB,UCBS%_DEVSTS,W,0 ; Internal status
0054 302 DPT_STORE DDB,DDBS%_DDT,D,NOSDDT ; Driver dispatch table
0059 303 DPT_STORE UCB,UCBS%_NO_AST,L,0 ; Attention AST listhead
0060 304 DPT_STORE UCB,UCBS%_DEVBUFSIZ,W,256 ; Default buffer size
0065 305 DPT_STORE UCB,UCBS%_DEVDEPEND,L,0 ; Clear devdepend field
006C 306 DPT_STORE UCB,UCBS%_DEVDEPND2,L,0 ; Clear devdepend field
0073 307 DPT_STORE ORB,ORBS%_FLAGS,B,- ; Protection block flags
0073 308 ZORBSM PROT_16> ; SOGW protection word
0077 309 DPT_STORE ORB,ORBS%_PROT,W,0 ; default protection
007C 310 DPT_STORE ORB,ORBS%_OWNER,L,<^X010001> ; [1,1] owns the device
0083 311 DPT_STORE UCB,UCBS%_STS,0
0087 312
0087 313 DPT_STORE REINIT ; Init data also for reload
0087 314 DPT_STORE CRB,CRBS%_INTD+VECS%_INITIAL,D,NOSCONTROL_INIT ; Controller init
008C 315 DPT_STORE CRB,CRBS%_INTD+VECS%_UNITINIT,D,NOSUNIT_INIT ; Unit init routine
0091 316 DPT_STORE END
0000 317

```



```
0000 319
0000 320 ; $QIO parameter offsets
0000 321
00000000 0000 322 P1 = 0 ; Parameter 1
00000004 0000 323 P2 = 4 ; Parameter 2
00000008 0000 324 P3 = 8 ; Parameter 3
0000000C 0000 325 P4 = 12 ; Parameter 4
00000010 0000 326 P5 = 16 ; Parameter 5
0000 327
0000 328 ; & temp until CLASS_LENGTH is define in TTYDEF.SDL
0000 329
00000028 0000 330 CLASS_LENGTH = CLASSSS_CLASS_DEF
0000 331
0000 332 ; & temp until defined in system
0000 333
000000C8 0000 334 DT$_ADDCMP = 200
0000 335
00000004 0000 336 DSC$A_POINTER = 4 ; Descriptor buffer address
00000004 0000 337 MAX_RCVS = 4 ; Max number of RCVS before
FFF3F300 0000 338 NO$C_LINE_PAR = ^X<FFF3F300> ; Bits to clear in DEVDEPEND
0000 339 ; on start of line
FFFFF700 0000 340 NO$C_CIR_PAR = ^X<FFFFF700> ; Bits to clear in DEVDEPEND
0000 341 ; on start of circuit
0000FFFF 0000 342 NO$C_PAD = ^X<FFFF> ; Pads to send at end of message
0000 343
0000 344 ; Bits set in this constant indicate the parity and the chaarcter length
0000 345 ; that we want the port driver to run at.
0000 346
00000018 0000 347 NO$C_SET_LINE = ^X<18>
0000 348
0000 349 ; NODRIVER constants
0000 350
0000 351 ; Message fields with constant values
0000 352
0000 353 CONSTDEF NO,ENQ,5 ; ENQ message identifier
0000 354 CONSTDEF NO,SOH,129 ; Data message identifier
0000 355 CONSTDEF NO,DLE,144 ; Maintenace message identifier
0000 356 CONSTDEF NO,SYN,150 ; SYN byte identifier
0000 357 CONSTDEF NO,DEL,255 ; DEL byte identifier
0000 358
0000 359 ; Receiver and transmitter states
0000 360
0000 361 CONSTDEF NO,SEAR_MSG,0 ; Receive state sear for start
0000 362 ; of message
0000 363 CONSTDEF NO,EMPTY,0 ; Tranmit state nothing being send
0000 364 CONSTDEF NO,HEADER,1 ; Send header; recv header+crc
0000 365 CONSTDEF NO,HCRC,2 ; Send header crc
0000 366 CONSTDEF NO,DATA,3 ; Send data; recv data+crc
0000 367 CONSTDEF NO,NOBUFFER,4 ; Recv no buffer for data
0000 368 CONSTDEF NO,DCRC,4 ; Send data crc
0000 369 CONSTDEF NO,PADS,5 ; Send PADS on end of message
0000 370 CONSTDEF NO,XMTERR,10 ; Send enough data to clear xmt erro
0000 371
0000 372 ; Misc constants
0000 373
0000 374 CONSTDEF NO,DEF_BUFSIZ,256 ; Default buffer size
0000 375 CONSTDEF NO,HEADER_LEN,DDCMP$C_HEADER ; Header length
```

```
0000 376 CONSTDEF NO,HEADER_HCRC,DDCMP$C HEADER+2 ; Header length plus crc length
0000 377 CONSTDEF NO,DUETIM_TABLE_SIZE,256 ; Default number of baud rates avail
0000 378
0000 379 ; Bit definitions
0000 380
0000 381 BITDEF NO_FS,PORTFORK,0 ; Fork on port
0000 382 BITDEF NO_FS,IOFORK,1 ; Fork for IO completion
0000 383 BITDEF NO_FS,POWERFAIL,2 ; Fork for powerfail completion
0000 384
0000 385 ; Overlay of IRP
0000 386
0000 387
0000 388 $DEFINI IRP
0000 389
00000021 0000 390 . = IRP$W_FUNC+1 ; Overlay function word
0021 391
0021 392 $DEF IRP$B_NOFUNC .BLKB 1 ; NODRIVER internal func code
0022 393
00000040 0022 394 . = IRP$Q_STATION
0040 395
0040 396 $DEF IRP$W_QUOTA .BLKW 1 ; NODRIVER quota used for receives
0042 397
0042 398 ; Define driver internal function codes stored in IRP$B_NOFUNC of IRP.
0042 399 ; NOTE: These are not used as bit offsets, but as values.
0042 400
0042 401 _VIELD NO_FC,0,<- ; Internal function codes
0042 402 <STRT_CIR>,- ; Start a tributary
0042 403 <STRT_LIN>,- ; Init the device inc UCB
0042 404 <STOP_CIR>,- ; Stop a tributary
0042 405 <STOP_LIN>,- ; Stop the device (inc stopping
0042 406 <READ_MODEM>,- ; Read the modem status
0042 407 >
0042 408
0042 409 $DEFEND IRP ; End of IRP overlays
0000 410
0000 411
0000 412 ; NODRIVER UCB OVERLAYS
0000 413
0000 414
0000 415 $DEFINI UCB
0000 416
000000A8 0000 417 . =UCB$Q_TL_BRKTHRU
00A8 418
00A8 419 ; Because we are limited to the size of the terminal driver UCB for our UCB
00A8 420 ; then when the device is started (the first QIO is issued to it) then we
00A8 421 ; will allocate a block from nonpaged pool which we will use as our UCB
00A8 422 ; extension. This block will be deallocate on the last DEASSIGN to the device.
00A8 423
00AC 424 $DEF UCB$L_NO_BUFFER .BLKL 1 ; Address of NOB buffer
00B0 425 $DEF UCB$L_NO_AST .BLKL 1 ; Attention AST list
00B0 426
00B0 427 $DEFEND UCB ; End of UCB overlays
0000 428
0000 429 ; These assumptions are made in case the terminal people decide to change
0000 430 ; the use of the UCB$W_DEVSTS field. Chagnes in bit definitions of that
0000 431 ; field could drastically affect the operation of this driver.
0000 432
```



```

0000 433      ASSUME  UCBSV_TT_TIMO EQ 1
0000 434      ASSUME  UCBSV_TT_NOTIF EQ 2
0000 435      ASSUME  UCBSV_TT_HANGUP EQ 3
0000 436      ASSUME  UCBSV_TT_NOLOGINS EQ 15
0000 437      ASSUME  UCBSV_TT_DEVSTS_FILL EQ 4
0000 438      ASSUME  UCBSS_TT_DEVSTS_FILL EQ 11
0000 439
0000 440      ; Device status bits
0000 441
00000000 0000 442      .=0
0000 443      _VIELD NO DS,4,<-      ; UCB$W_DEVSTS bits
0000 444      <XMTING,,M>,-      ; Device XMTing if 0
0000 445      <RCVING,,M>,-      ; Device RCVing if 0
0000 446      <INITED,,M>,-      ; Unit initialized
0000 447      <FORK PEND,,M>,-      ; Fork pending
0000 448      <ILOOP SUP,,M>,-      ; Internal loop supported on device
0000 449      <MSG SENT,,M>,-      ; Set when a msg was xmted
0000 450      <SHUTDOWN,,M>,-      ; Set when circuit going thru shutdo
0000 451      <XMT_TIME,,M>,-      ; Set if XMT_DUETIM is valid
0000 452      <RCV_TIME,,M>,-      ; Set if RCV_DUETIM is valid
0000 453      >
0000 454
0000 455      ; Receive Flag field bit definitions
0000 456
0000 457      BITDEF          RCV,CNTL,0      ; Set if rcv msg is an ENQ msg
0000 458
0000 459      ; Receive buffer definition
0000 460      ;
0000 461      ; This structure must be the same size as the CXB
0000 462
00000000 0000 463      $DEFINI RCV
0000 464      . = 0
0000 465      $DEF          RCV_L_LINK          .BLKL      2      ; Forward and backward links
0000 466      $DEF          RCV_W_BLKSIZE      .BLKW      1      ; Total block size
000A 467      $DEF          RCV_B_BLKTYPE      .BLKB      1      ; Block type
000B 468      $DEF          RCV_B_FIPL        .BLKB      1      ; Fork IPL
000C 469      $DEF          RCV_W_MSGSIZ      .BLKW      1      ; Size of this part of the msg to r
000E 470      $DEF          RCV_W_INDEX        .BLKW      1      ; Index into rcved message
0010 471      $DEF          RCV_W_ERROR        .BLKW      1      ; Error status
0012 472      $DEF          RCV_W_FLAGS        .BLKW      1      ; Receive message flags
0014 473      $DEF          RCV_CXB_SPARE      .BLKB      CXB$K_HEADER-NOSC-HEADER_HCRC-      ; Spare bytes to allow for CXB
0040 474      ;
0040 475      $DEF          RCV_Z_HEADER          .BLKB      NOSC_HEADER_HCRC      ; Size in bytes of msg header
0048 476      $DEF          RCV_T_DATA          ; Receive data
0048 477
0048 478      ASSUME  RCV_Z_HEADER+8 EQ CXB$K_HEADER
0048 479      ASSUME  RCV_T_DATA GE CXB$K_HEADER
0048 480
0048 481      $DEFEND RCV
0000 482
0000 483      ; P2 buffer header definition
0000 484
00000000 0000 485      $DEFINI P2B
0000 486      .=0
0000 487      $DEF          P2B_L_POINTER          .BLKL      1      ; Pointer to start of data
0004 488      $DEF          P2B_L_BUFFER          .BLKL      1      ; Address of user's data buffer
000B 489      $DEF          P2B_W_SIZE          .BLKW      1      ; Size of P2 buffer

```

```
000A 490 $DEF P2B_B_TYPE .BLKB 1 ; Type of structure
000B 491 $DEF P2B_B_SPARE .BLKB 1 ; Spare byte
000C 492 $DEF P2B_C_LENGTH ; Size of P2 buffer header
000C 493 $DEF P2B_T_DATA ; Start of data
000C 494
000C 495 $DEFEND P2B
0000 496
0000 497
0000 498 ; NODRIVER buffer definitions
0000 499
0000 500 $DEFINI NOB
00000000 0000 501 .=0
0000 502 $DEF NOBSL_FLINK .BLKL ; Nob buffer forward link
0004 503 $DEF NOBSL_BLINK .BLKL ; Nob buffer backward link
0008 504 $DEF NOBSW_SIZE .BLKW ; Size of Nob
000A 505 $DEF NOBSB_TYPE .BLKB ; Type of buffer
0000000C 000B 506 .BLKB ; reserved
000C 507 $DEF NOBSA_PRO BUFFER .BLKL 1 ; Address of the DDCMP buffer
0010 508 $DEF NOBSQ_ATTQ .BLKQ 1 ; Received message list
0018 509 $DEF NOBSQ_RCVS .BLKQ 1 ; Received I/O queue
0020 510 $DEF NOBSQ_FREE .BLKQ 1 ; Free receive buffer queue
0028 511 $DEF NOBSQ_POST .BLKQ 1 ; Receive and transmit buffers to comp
0030 512 $DEF NOBSL_RCV_INPR .BLKL 1 ; Receive buffer awaiting data
0034 513 $DEF NOBSL_XMT_INPR .BLKL 1 ; Transmit buffer to be sent
0038 514 $DEF NOBSL_RCV_DUETIME .BLKL 1 ; Receive message duetime
003C 515 $DEF NOBSL_XMT_DUETIME .BLKL 1 ; Transmit message duetime
0040 516 $DEF NOBSW_QUOTA .BLKW 1 ; Byte quota for receive's
0042 517 $DEF NOBSB_XSTATE .BLKB 1 ; Transmitter state
0043 518 $DEF NOBSB_RSTATE .BLKB 1 ; Receiver state
0044 519 $DEF NOBSW_PADS .BLKW 1 ; Pads to send at end of messages
0046 520 $DEF NOBSW_XMTERR_SIZE .BLKW 1 ; Number of pads to send on an error on an x
0048 521 $DEF NOBSZ_HEADER .BLKB 1 ; NO$C_HEADER_HCRC ; Area to receive header and crc
0050 522 $DEF NOBSW_MSGSIZ .BLKW 1 ; Current number of bytes left in rec hdr
0052 523 $DEF NOBSW_INDEX .BLKW 1 ; Index into receive header
0054 524 $DEF NOBSW_ERROR .BLKW 1 ; Error on receive if any
0056 525 $DEF NOBSW_FLAGS .BLKW 1 ; Flags for receive buffer after it is alloc
0058 526 $DEF NOBSL_PID .BLKL 1 ; Process ID
005C 527 $DEF NOBSW_CHANL .BLKW 1 ; Line channel number
005E 528 $DEF NOBSW_CHANC .BLKW 1 ; Circuit channel number
0060 529 $DEF NOBSZ_DDCMP .BLKL 4 ; Block for setable DDCMP parameters
0070 530 $DEF NOBS_C_SETPRM ; Start of UCB/LINE paramters
0070 531 $DEF NOBSB_PRO .BLKB 1 ; Protocol selection
0071 532 $DEF NOBSB_DUP .BLKB 1 ; Duplex setting
0072 533 $DEF NOBSB_CON .BLKB 1 ; Controller loopback
0073 534 $DEF NOBSB_BFN .BLKB 1 ; Number of receive buffers
0074 535 $DEF NOBSW_DEVBUSIZ .BLKW 1 ; Size of receive buffers
0076 536 $DEF NOBSB_SPD .BLKB 1 ; Set line speed
00000078 0077 537 .BLKB 1 ; reserved
0078 538 $DEF NOBSZ_DLA_ADDR .BLKB 1 ; DLASC_ADDR_LENGTH ; Buffer to pass to prot with
0080 539 ; ; address of queues prot needs
0080 540
0080 541 ; Block to copy the control message into on transmit
0080 542 $DEF NOBSZ_CTL_MSG .BLKB 1 ; XMTQ$K_LENGTH
00AA 543
00AA 544 $DEF NOBS_C_LENGTH ; Length of the NODRIVER buffer
00AA 545 $DEF NOBSK_LENGTH ; Length of the NODRIVER buffer
00AA 546
```


NODRIVER
V04-001

```
00AA 547 $DEFEND NOB
00000000 548 .PSECT $$$115_DRIVER, LONG
0000 549
0000 550
0000 551 ; Driver Dispatch Table
0000 552
0000 553 DDTAB DEVNAM=NO,- ; Device name
0000 554 START=NO$STARTIO,- ; Start I/O routine
0000 555 FUNCTB=NO$FUNCTABLE,- ; Function decision table
0000 556 CANCEL=NO$CANCEL,- ; Cancel I/O routine
0000 557 REGDMP=NO$REGDUMP,- ; Register dump routine
0000 558 DIAGBF=<36+12>,- ; Diagnostic buffer size
0000 559 ALTSTART=NO$ALT_ENTRY ; Alternate entry routine
```

```

0038 561
0038 562 ; Function Decision Table
0038 563
0038 564 NOSFUNCTIONTABLE:
0038 565     FUNCTAB,-
0038 566         <WRITEVBLK,WRITELBLK,WRITEPBLK,-; Legal functions
0038 567         READVBLK,READLBLK,READPBLK,-; Transmit functions
0038 568         SETMODE,SENSEMODE,SETCHAR,-; Receive functions
0038 569         >; Set mode functions
0040 570     FUNCTAB,-
0040 571         <WRITEVBLK,WRITELBLK,WRITEPBLK,-; Buffered I/O functions
0040 572         READLBLK,READPBLK,READVBLK,-; Transmit functions
0040 573         SETMODE,SENSEMODE,SETCHAR,-; Receive functions
0040 574         >; Set mode functions
0048 575
0048 576 .IF DF JNX$$$
0048 577     functab fillbuffer,-; Buffered I/O functions
0048 578         <writevblk,writelblk,writepblk,-; Transmit functions
0048 579         readvblk,readlblk,readpblk,-; Receive functions
0048 580         setmode,sensemode,setchar,-; Set mode functions
0048 581         >
0048 582 .ENDC ;DF JNX$$$
0048 583
0048 584 FUNCTAB NOSXMITFDT,-; Transmit function dispatcher
0048 585     <WRITELBLK,WRITEPBLK,WRITEVBLK>
0054 586 FUNCTAB NOSRCVFDT,-; Receive function dispatcher
0054 587     <READLBLK,READPBLK,READVBLK>
0060 588 FUNCTAB NOSSETMODEFDT,-; FDT for set mode and set char
0060 589     <SETMODE,SETCHAR>
006C 590 FUNCTAB NOSSENSEMODEFDT,-; FDT sensemode routine
006C 591     <SENSEMODE>
0078 592

```



```
0078 594
0078 595 : The following is a table of services that the ASYNCH DDCMP driver uses to
0078 596 : interface with the terminal lines' port driver. It initially contains
0078 597 : relative offsets to various routines and data structures needed by the
0078 598 : terminal port driver. At driver load these relative offsets are reloacted
0078 599 : to actual virtual addresses. The list is terminated by a 0 longword to
0078 600 : signal te relocation routine where the list terminates.
0078 601 :
0078 602 : NOTE the because of how this table is indexed into the following order
0078 603 : can not be changed, and if the placement of the routines are changed we
0078 604 : must know about it.
0078 605
0078 606 ASSUME CLASS_GETNXT EQ 0
0078 607 ASSUME CLASS_PUTNXT EQ 4
0078 608 ASSUME CLASS_SETUP_UCB EQ 8
0078 609 ASSUME CLASS_DS_TRAN EQ 12
0078 610 ASSUME CLASS_DDT EQ 16
0078 611 ASSUME CLASS_READERROR EQ 20
0078 612 ASSUME CLASS_DISCONNECT EQ 24
0078 613 ASSUME CLASS_FORK EQ 28
0078 614 ASSUME CLASS_POWERFAIL EQ 32
0078 615
0078 616 ADDCMP_VECTOR:
000010FA' 0078 617 .LONG NOSGETNXT - NOSDPT : Port driver calls to get next
007C 618 : character to output
00001274' 007C 619 .LONG NOSPUTNXT - NOSDPT : Port driver calls whenever a
0080 620 : character is received
000013E8' 0080 621 .LONG NOSSETUP_UCB - NOSDPT : Called to reset units' UCB,
0084 622 : called at powerfail and unit init
000013E9' 0084 623 .LONG NOSPORT_TRANSITION - NOSDPT : Handles modem transitions
0088 624 : on the line
00000000' 0088 625 .LONG NOSDDT - NOSDPT : Class drivers DDT
000013EA' 008C 626 .LONG NOSREADERROR - NOSDPT : Port driver detected an error
0090 627 : on the line
00001449' 0090 628 .LONG NOSCLASS_DISCONNECT - NOSDPT : Indicates that the terminal
0094 629 : is no longer connect to the system
000013A0' 0094 630 .LONG NOSCLASS_PORTFORK - NOSDPT : Used when the port driver
0098 631 : needs to issue a fork
0000144A' 0098 632 .LONG NOSCLASS_POWERACTION - NOSDPT : Called in unit init when a
009C 633 : powerfail occured
00000000 009C 634 .LONG 0
```

```
00A0 636 .SBTTL P2 buffer verification tables
00A0 637
00A0 638
00A0 639 ; Define P2 buffer verification offsets
00A0 640
00A0 641 $DEFINI PARAM
0000 642
0000 643 _VIELD PRM,0,<- ; Parameter bits and sizes
0000 644 <TYPE,12,M>,- ; Parameter type
0000 645 <MIN,1,M>,- ; Parameter minimum value
0000 646 <MAX,1,M>,- ; Parameter maximum value
0000 647 <INVALID,1,M>,- ; Parameter invalid flags
0000 648 >
0000 649
0000 650 _VIELD OFF,0,<- ; Offset word fields
0000 651 <VALUE,14,M>,- ; Offset value
0000 652 <WIDTH,2,M>,- ; Size of field in structure
0000 653 >
0000 654
0000 655 $DEFEND PARAM
00A0 656
00A0 657
00A0 658 ; Define UCB (line) parameters
00A0 659
00000000 00A0 660 LINE_PRM_BUFSIZ = 0 ; Line parameter buffer size
00A0 661 LINE_PARAM: ; Start of line parameters
00A0 662
00A0 663 PARAM NMASC_PCLI_PRO,- ; Protocol selection
00A0 664 OFFSET=NOB$B_PRO,WIDTH=B,-
00A0 665 MAX=NMASC_LINPR_TRI,-
00A0 666 INVALID=NO_DS_M_INITED,- ; Device can't be ON
00A0 667 BASE=LINE
00A8 668
00A8 669 PARAM NMASC_PCLI_DUP,- ; Duplex mode
00A8 670 OFFSET=NOB$B_DUP,WIDTH=B,-
00A8 671 MAX=NMASC_DPR_HAL,-
00A8 672 INVALID=NO_DS_M_INITED,- ; Device can't be ON
00A8 673 BASE=LINE
00B0 674
00B0 675 PARAM NMASC_PCLI_CON,- ; Controller mode
00B0 676 OFFSET=NOB$B_CON,WIDTH=B,-
00B0 677 MAX=NMASC_LINCN_LOO,-
00B0 678 INVALID=NO_DS_M_INITED,- ; Device can't be On
00B0 679 BASE=LINE
00B8 680
00B8 681 PARAM NMASC_PCLI_BFN,- ; Number of receives
00B8 682 OFFSET=NOB$B_BFN,WIDTH=B,-
00B8 683 MIN=1,MAX=255,-
00B8 684 INVALID=NO_DS_M_INITED,- ; Device can't be ON
00B8 685 BASE=LINE
00C2 686
00C2 687 PARAM NMASC_PCLI_BUS,- ; Buffer size
00C2 688 OFFSET=NOB$W_DEVBUSIZ,WIDTH=W,-
00C2 689 MIN=1,MAX=16383,-
00C2 690 INVALID=NO_DS_M_INITED,- ; Device can't be ON
00C2 691 BASE=LINE
00CC 692
```


- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 17
P2 buffer verification tables 6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (14)

[illegible]


```
00F4 732
00F4 733
00F4 734 ; DEFAULT TRIBUTARY PARAMETERS for DDCMP
00F4 735
00F4 736 DEF_TRIB_PARAM::
00F4 737     ASSUME DLK$B_MSGCNT EQ DLK$B_TRIB+1
00F4 738     ASSUME DLK$B_MAXREP EQ DLK$B_MSGCNT+1
00F4 739     ASSUME DLK$B_MAXSEL EQ DLK$B_MAXREP+1
00F4 740     ASSUME DLK$W_REPWAIT EQ DLK$B_MAXSEL+1
00F4 741     ASSUME DLK$W_SELWAIT EQ DLK$W_REPWAIT+2
00F4 742     ASSUME DLK$B_MAINT EQ DLK$W_SELWAIT+2
00F4 743     ASSUME DLK$B_MRB EQ DLK$B_MAINT+1
01 00F4 744     .BYTE 1 ; Default Trib address
04 00F5 745     .BYTE 4 ; Max nmb of msgs sent / select
04 00F6 746     .BYTE 4 ; Max nmb of sel intls allowed
03 00F7 747     .BYTE 3 ; Max times to reXMT a msg
0BB8 00F8 748     .WORD 3000 ; Reply timeout timer
0BB8 00FA 749     .WORD 3000 ; Selection timer in sec
01 00FC 750     .BYTE NMA$C_STATE_OFF ; Default is no maint mode
FF 00FD 751     .BYTE 255 ; Default to unlimited rcv buff
0000000A 00FE 752     DEF_TRIB_PARAMSZ = .-DEF_TRIB_PARAM
00FE 753
00FE 754 ; Default line parameter values
00FE 755
00FE 756 DEF_LINE_PARAM::
00FE 757     ASSUME NOB$B_PRO EQ NOB$C_SETPRM
00FE 758     ASSUME NOB$B_DUP EQ NOB$B_PRO+1
00FE 759     ASSUME NOB$B_CON EQ NOB$B_DUP+1
00FE 760     ASSUME NOB$B_BFN EQ NOB$B_CON+1
00FE 761     ASSUME NOB$W_DEVBUSIZ EQ NOB$B_BFN+1
00FE 762
00 00FE 763     .BYTE NMA$C_LINPR_POI ; Protocol is point-point
00 00FF 764     .BYTE NMA$C_DPX_FOL ; Duplex is full
00 0100 765     .BYTE NMA$C_LINC_NOR ; Controller mode is normal
01 0101 766     .BYTE 1 ; Number of receive buffers
0100 0102 767     .WORD NO$C_DEF_BUFSIZ ; Size of receive buffers
0104 768
00000006 0104 769     DEF_LINE_PARAMSZ = .-DEF_LINE_PARAM
0104 770
0104 771 ; Table to store all possible duetimes for receive and tranmsit messages.
0104 772 ; The time to wait is store in this table and is calculated as follows:
0104 773 ; ((default buffer size + the ddcmp overhead bytes) * eight bits per byte )
0104 774 ; divided the baud rate. This table is calculated each time the line is
0104 775 ; started so we will have the most recent information about the default buffer
0104 776 ; size. The byte value of the terminal line speed is used to index into this
0104 777 ; table.
0104 778
00000304 0104 779 DUETIM_TABLE:
0104 780     .BLKW NO$C_DUETIM_TABLE_SIZE
0304 781
```



```

0304 783 .SBTTL NOSCONTROL_INIT - Initialize Sync line device
0304 784 :++
0304 785 : NOSCONTROL_INIT - Initialize Sync line unit
0304 786 :
0304 787 : FUNCTIONAL DESCRIPTION:
0304 788 :
0304 789 : This routine is called at driver load it sets up the global DPT for the
0304 790 : driver adn relocates the drivers vector tables.
0304 791 :
0304 792 : INPUTS:
0304 793 : R4 = Address of device CSR
0304 794 : R5 = Address of device IDB
0304 795 : R6 = Address of device DDB
0304 796 : R8 = Address of device CRB
0304 797 :
0304 798 : OUTPUTS:
0304 799 : R4,R5,R8 are preserved
0304 800 :--
0304 801 NOSCONTROL_INIT:
0304 802 MOVAL NOSDPT,R1
0308 803 MOVL R1,G^NOSGL_DPT
0312 804
0312 805 MOVZWL DPT$W_VECTOR(R1),R0
0316 806 ADDL2 R1,R0
0319 807
0319 808 ; Relocate the asynch ddcmp vector table
0319 809
0319 810 5$: TSTL (R0)
0318 811 BLEQ 10$
031D 812 ADDL2 R1,(R0)+
0320 813 BRB 5$
0322 814
0322 815 10$: RSB
0323 816

```

51 00000000'EF DE 0304 802
 00000000'GF 51 D0 0308 803
 50 1E A1 3C 0312 804
 50 51 C0 0316 805
 60 D5 0319 806
 05 15 0319 807
 80 51 C0 0318 808
 F7 11 031D 809
 05 0320 810
 0322 811
 0323 812

```
0323 818 .SBTTL NOSUNIT_INIT - Initialize the device unit
0323 819 :++
0323 820 : NOSUNIT_INIT - Initialize the device unit
0323 821 :
0323 822 : FUNCTIONAL DESCRIPTION:
0323 823 :
0323 824 : This routine is called when the driver is loaded and during powerfailure
0323 825 : recovery. It sets the unit status to ONLINE. Also, if called during
0323 826 : powerfail recovery, it shuts down the device.
0323 827 :
0323 828 : INPUTS:
0323 829 : R4 = Address of the device CSR
0323 830 : R5 = UCB address
0323 831 :
0323 832 : OUTPUTS:
0323 833 : R5 preserved
0323 834 :--
0323 835 NOSUNIT_INIT:
0323 836 BBC #UCBSV_POWER,UCBSW_STS(R5),10$ ; Initialize the unit
0323 837 BBC #XMSV_STS_ACTIVE,- ; Br if not powerfail recovery
0323 838 UCB$L_DEVDEPEND(R5),10$ ; Br if not previously active
0323 839 PUSHF #^M<R0,R1,R2,R3> ; Save all registers
0323 840 BSBW SCHED_FORK_POWERFAIL
0323 841 POPR #^M<R0,R1,R2,R3>
0323 842 RSB
0323 843
0323 844 5$: BUG_CHECK NOBUFCKT,FATAL
0323 845
0323 846
0323 847 : This driver makes the assumption that IPL$_SYNCH, IPL$_TIMER and the drivers
0323 848 : fork IPL are all equal. If any of these change the integrity of the driver
0323 849 : can not be assured. THIS DRIVER WILL NOT WORK ON PRE VERSION 4 SYSTEMS.
0323 850
0323 851
0323 852 ASSUME IPL$_SYNCH EQ IPL$_TIMER
0323 853
0323 854
0323 855 : We must also be sure that SYNCH and the drivers FIPL are the same. If this
0323 856 : changes the integrity of the driver can not be assured with out some
0323 857 : major changes.
0323 858
0323 859 10$: CMPB #IPL$_SYNCH,UCBSB_FIPL(R5) ; If the IPLs are not equal then
0323 860 BNEQ 5$ ; cause a fatal bugcheck
0323 861 BISW #UCBSM_ONLINE,UCBSW_STS(R5) ; Set software status ONLINE
0323 862 RSB
0323 863
```

11 64 A5 05 E1
OC 44 A5 0B E1
OF BB
109E 30
OF BA
05 05

08 A5 08 91
F6 12
64 A5 10 A8
05 05

NODRIVER
V04-001

F 14
- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 21
NOSUNIT_INIT - Initialize the device uni 6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (19)

```
0344 865 .IF DF JNX$$$  
0344 866 fillbuffer:  
0344 867 movzbl ucb$b_last(r5),r0  
0344 868 movl r3,ucb$l_buffer(r5)[r0]  
0344 869 movq #0,irp$l_media(r3)  
0344 870 incl r0  
0344 871 bicl #^c<jnx_size-1>,r0  
0344 872 movb r0,ucb$b_last(r5)  
0344 873 rsb  
0344 874 .ENDC ;DF JNX$$$  
0344 875
```

```
0344 877 .SBTTL NOSXMITFDT - Transmit I/O FDT routine
0344 878 :++
0344 879 : NOSXMITFDT - Transmit I/O FDT routine
0344 880 :
0344 881 : FUNCTIONAL DESCRIPTION:
0344 882 :
0344 883 : This routine allocates a system buffer for the QIO. Then calls the
0344 884 : common FDT routine give the buffer to DDCMP.
0344 885 :
0344 886 : INPUTS:
0344 887 : R3 = I/O packet address
0344 888 : R4 = Current PCB address
0344 889 : R5 = UCB address
0344 890 : R6 = CCB address
0344 891 :
0344 892 : OUTPUTS:
0344 893 : R3,R4,R5,R6,R7,R8,R9 are preserved
0344 894 :
0344 895 :--
0344 896 NOSXMITFDT:
0344 897 PUSH R3,R4,R5,R6,R7,R8,R9 : Transmit FDT routine
0344 898 MOV P1(AP),R7 : Get address of buffer
0344 899 MOV P2(AP),R9 : Get the size of buffer
0344 900 BEQL 10$ : If EQL then size is zero
0344 901 MOV R7,R0 : Set up R0 and R1 for
0344 902 MOV R9,R1 : write access check
0344 903 JSB G^EXE$WRITECHK : Check the acc of users buffer
0344 904 ADDL3 #XMTQ$K_LENGTH,R9,R1 : Get length of buffer to alloc
0344 905 MOV R3,R6 : Save the IRP address
0344 906 JSB G^EXE$BUFFRQUOTA : Check quota
0344 907 BLBC R0,20$ : If LBC not enough quota
0344 908 JSB G^EXE$ALLOCBUF : Allocated the buffer
0344 909 BLBC R0,20$ : If LBC not buffer allocated
0344 910 MOV R6,R3 : Retrieve the IRP
0344 911 MOV R2,R8 : Get buffer address in R8
0344 912 MOV PCB$JIB(R4),R0 : Get the JIB address
0344 913 SUBL2 R1,JIB$BYTCNT(R0) : Adjust buffered I/O
0344 914 MOVW R1,IRP$W_BOFF(R3) : Save byte offset
0344 915 MOV R2,IRP$SVAPTE(R3) : Save system PTE
0344 916 CLRB XMTQ$FLAG(R8) : Clr bits, not "Internal" IRP
0344 917 5$: BSBW COP_BUFF : Branch to copy into my buff
0344 918 SETIPL UCB$F IPL(R5) : Sync to FIPL
0344 919 BSBB COM_XMITFDT : Branch to common processing
0344 920 BLBC R0,20$ : If BC then error
0344 921 8$: POP R3,R4,R5,R6,R7,R8,R9
0344 922 JMP G^EXE$QIORETURN
0344 923
0344 924 10$: MOVZWL S^#SS$_BADPARAM,R0 : Set status for abort
0344 925 20$: POP R3,R4,R5,R6,R7,R8,R9
0344 926 JMP G^EXE$ABORTIO
0344 927
```

03F8 8F BB	0344 897	PUSH R3,R4,R5,R6,R7,R8,R9	: Transmit FDT routine
57 6C DO	0344 898	MOV P1(AP),R7	: Get address of buffer
59 04 AC DO	0344 899	MOV P2(AP),R9	: Get the size of buffer
55 13	0344 900	BEQL 10\$	: If EQL then size is zero
50 57 DO	0344 901	MOV R7,R0	: Set up R0 and R1 for
51 59 DO	0344 902	MOV R9,R1	: write access check
00000000'GF 16	0344 903	JSB G^EXE\$WRITECHK	: Check the acc of users buffer
51 59 2A C1	0344 904	ADDL3 #XMTQ\$K_LENGTH,R9,R1	: Get length of buffer to alloc
56 53 DO	0344 905	MOV R3,R6	: Save the IRP address
00000000'GF 16	0344 906	JSB G^EXE\$BUFFRQUOTA	: Check quota
3C 50 E9	0344 907	BLBC R0,20\$	: If LBC not enough quota
00000000'GF 16	0344 908	JSB G^EXE\$ALLOCBUF	: Allocated the buffer
33 50 E9	0344 909	BLBC R0,20\$	: If LBC not buffer allocated
53 56 DO	0344 910	MOV R6,R3	: Retrieve the IRP
58 52 DO	0344 911	MOV R2,R8	: Get buffer address in R8
50 0080 C4 DO	0344 912	MOV PCB\$JIB(R4),R0	: Get the JIB address
20 A0 51 C2	0344 913	SUBL2 R1,JIB\$BYTCNT(R0)	: Adjust buffered I/O
30 A3 51 B0	0344 914	MOVW R1,IRP\$W_BOFF(R3)	: Save byte offset
2C A3 52 DO	0344 915	MOV R2,IRP\$SVAPTE(R3)	: Save system PTE
1F A8 94	0344 916	CLRB XMTQ\$FLAG(R8)	: Clr bits, not "Internal" IRP
0076 30	0344 917	5\$: BSBW COP_BUFF	: Branch to copy into my buff
1A 10	0344 918	SETIPL UCB\$F IPL(R5)	: Sync to FIPL
0D 50 E9	0344 919	BSBB COM_XMITFDT	: Branch to common processing
03F8 8F BA	0344 920	BLBC R0,20\$	: If BC then error
00000000'GF 17	0344 921	8\$: POP R3,R4,R5,R6,R7,R8,R9	
50 14 3C	0344 922	JMP G^EXE\$QIORETURN	
03F8 8F BA	0344 923		
00000000'GF 17	0344 924	10\$: MOVZWL S^#SS\$_BADPARAM,R0	: Set status for abort
	0344 925	20\$: POP R3,R4,R5,R6,R7,R8,R9	
	0344 926	JMP G^EXE\$ABORTIO	
	0344 927		


```
03B3 929 .SBTTL COM_XMITFDT - Common transmit FDT routine
03B3 930 :++
03B3 931 :COM_XMITFDT - Common transmit FDT routine
03B3 932 :
03B3 933 : The allocated buffer is given to DDCMP where a header is added and if
03B3 934 : possible the buffer is added to the transmit queue. The buffer keeps various
03B3 935 : information on the transmit as it progresses from the transmit to the
03B3 936 : completion queues.
03B3 937 :
03B3 938 : INPUTS:
03B3 939 : R3 = IRP address
03B3 940 : R5 = UCB address
03B3 941 : R7 = Address of the user/"Internal" IRP buffer
03B3 942 : R8 = Address of allocated buffer
03B3 943 : R9 = Size of user/"Internal" IRP buffer
03B3 944 :
03B3 945 : IPL = Fork IPL
03B3 946 :
03B3 947 : OUTPUTS:
03B3 948 : R0 = Status of operation
03B3 949 : R5,R8 are preserved
03B3 950 :
03B3 951 :--
03B3 952 COM_XMITFDT:
54 00A8 C5 D0 03B3 953 MOVL UCBSL_NO_BUFFER(R5),R4 ; Get NOB address
    45 13 03B8 954 BEQL 45$ ; If eql device not started
    30 BB 03BA 955 PUSHR #M<R4,R5> ; Save these registers
55 0C A4 D0 03BC 956 MOVL NOBSA_PRO_BUFFER(R4),R5 ; Get addr of start of TFB
    3B 13 03C0 957 BEQL 43$
    56 02 9A 03C2 958 MOVZBL #DLKSC_XMTMSG,R6 ; Set that this is a msg to XMT
    57 04 9A 03C5 959 CLRL R7 ; Clear error bits
    FC36' 30 03C7 960 BSBW DDCMP ; Branch to set up the header
    30 BA 03CA 961 POPR #M<R4,R5> ; Restore the registers
    56 09 91 03CC 962 CMPB #DLKSC_ACTNOTCOM,R6 ; If EQL then protocol has not
    2E 13 03CF 963 BEQL 45$ ; been started
    1C 57 04 E0 03D1 964 BBS #DLKSV_XMTERR,R7,35$ ; If BS then problem with XMT
    03 57 0A E1 03D5 965 BBC #DLKSV_XMTCMP,R7,20$ ; If BC no XMT's to complete
    1233 30 03D9 966 BSBW FINISH_XMT IO ; Complete all XMT's
    16 57 01 E0 03DC 967 20$: BBS #DLKSV_PRSTERR,R7,40$ ; Branch BS persisent error
57 0220 8F B3 03E0 968 BITW #<DLKSM_XMTACK!- ; If EQL then XMT not put on
    0A 13 03E5 969 DLKSM_QFULERR>,R7 ; either queue so abort it
    51 D4 03E7 970 BEQL 35$
    085B 30 03E9 971 CLRL R1 ; Set no status for start_transmit
    50 01 3C 03EC 972 BSBW START_TRANSMIT ; Else start the xmt'r
    13 11 03EF 973 30$: MOVZWL S^SS$_NORMAL,R0 ; Set normal return
    03F1 974 BRB 50$
    50 14 3C 03F1 975 35$: MOVZWL S^SS$_BADPARAM,R0 ; Set status
    0E 11 03F4 976 BRB 50$
    03F6 977
    EF 11 03F6 978 SETBIT #XMSV_ERR_FATAL,UCBSL_DEVDEPEND(R5) ; Set fatal error
    03FB 979 40$: BRB 30$ ; Complete as normal timer
    03FD 980 ; will shutdown the device
    30 BA 03FD 981 POPR #M<R4,R5> ; Restore the registers
50 20D4 8F 3C 03FF 982 43$: MOVZWL #SS$ DEVINACT,R0 ; Set the device is not active
    51 44 A5 D0 0404 983 45$: MOVL UCBSL_DEVDEPEND(R5),R1 ; Set status for abort
    05 0408 984 50$: RSB
    985
```

```
0409 986
0409 987 :++
0409 988 :COP_BUFF
0409 989
0409 990 : This routine takes the user or "Internal" IRP buffer which contains the
0409 991 : message to send and writes it into the allocated buffer.
0409 992
0409 993 : INPUTS:
0409 994 : R1 = Size of allocated buffer
0409 995 : R3 = IRP address
0409 996 : R8 = Address of allocated buffer
0409 997 : R7 = Address of buffer from which to move data
0409 998 : R9 = Size of user/"Internal" IRP buffer
0409 999
0409 1000 : IPL = FIPL from ALT_ENTRY and ASTDEL from XMITFDT
0409 1001
0409 1002 : OUTPUTS:
0409 1003 : Buffer is copied into allocated buffer
0409 1004 : R5,R8,R9 are preserved
0409 1005
0409 1006 :--
0409 1007 :COP_BUFF:
0409 1008 : MOVW R1,XMTQ$W BUFLen(R8) ; Save the size of the buffer
0409 1009 : MOV B S^#DYN$C BUFIO,- ; Set that this is an XMT
0409 1010 : XMTQ$B BOFTYP(R8)
0409 1011 : MOVL R3,XMTQ$L IRP(R8) ; Save address of the IRP
0409 1012 : ADDW3 #MFD$K_LENGTH,R9,- ; Get the msg size plus header
0409 1013 : XMTQ$W_MSGSIZE(R8) ; for character count
0409 1014 : PUSH R5 ; Save R5 before the MOV
0409 1015 : MOVC R9,(R7),XMTQ$K_LENGTH(R8) ; Move data into system buffer
0409 1016 : POPL R5 ; Restore R5
0409 1017 : RSB
```

08 A8	51	B0	0409	1008	MOVW	R1,XMTQ\$W BUFLen(R8)	; Save the size of the buffer
	13	90	040D	1009	MOVB	S^#DYN\$C BUFIO,-	; Set that this is an XMT
	0A A8		040F	1010		XMTQ\$B BOFTYP(R8)	
OC A8	53	D0	0411	1011	MOVL	R3,XMTQ\$L IRP(R8)	; Save address of the IRP
1A A8	59	A1	0415	1012	ADDW3	#MFD\$K_LENGTH,R9,-	; Get the msg size plus header
			041A	1013		XMTQ\$W_MSGSIZE(R8)	; for character count
	55	DD	041A	1014	PUSHL	R5	; Save R5 before the MOV
2A A8	67	59	041C	1015	MOVC	R9,(R7),XMTQ\$K_LENGTH(R8)	; Move data into system buffer
	55	8ED0	0421	1016	POPL	R5	; Restore R5
		05	0424	1017	RSB		


```
0425 1019 .SBTTL NOSALT_ENTRY - Alternate I/O entry
0425 1020 :++
0425 1021 : NOSALT_ENTRY - Alternate I/O entry point
0425 1022 :
0425 1023 : This routine is called by the other drivers to pass an "internal" I/O
0425 1024 : request to the driver. "Internal" IRP's are not built via $QIO.
0425 1025 : The action here is to setup the IRP fields as if the packet had been
0425 1026 : processed by the FDT routines.
0425 1027 :
0425 1028 : In this driver, the alternate entry point is called by the DECnet
0425 1029 : Transport layer driver.
0425 1030 :
0425 1031 : INPUTS:
0425 1032 : R3 = IRP address
0425 1033 : R5 = UCB address
0425 1034 : All pertinent fields of the IRP are assumed to be valid.
0425 1035 :
0425 1036 : IPL = Fork IPL
0425 1037 :
0425 1038 : OUTPUTS:
0425 1039 : R0 = Status of the request
0425 1040 : R3 and R5 preserved
0425 1041 :--
0425 1042 NOSALT_ENTRY: ; Accept an "internal" IRP
0425 1043 .IF DF JNX$$$
0425 1044 bsbw fillbuffer
0425 1045 .ENDC ;DF jnx$$$
0425 1046
0425 1047 MOVZWL #$$$ DEVINACT,IRP$L_MEDIA(R3) ; Assume device inactive
0425 1048 BBS #XMSV_STS_ACTIVE,- ; If BS status active
0425 1049 UCB$L_DEVDEPEND(R5),5$ ; device is on line
0425 1050 BRW IO_DONE ; Else comp request in error
0425 1051 BBS #IRPSV_FUNC,- ; If BS then receive function
0425 1052 IRPSV_STS(R3),ALT_RCVFDT
0425 1053 #*M<R3,R4,R5,R6,R7,R8,R9> ; Save registers
0425 1054 PUSHB #*M<R3,R4,R5,R6,R7,R8,R9> ; Get the length of the buffer
0425 1055 MOVZWL IRPSV_BCNT(R3),R9 ; If EQL then bad parameter
0425 1056 BEQL 10$ ; Add the XMTQ length
0425 1057 ADDL3 R9,#XMTQSK_LENGTH,R1 ; Save the IRP
0425 1058 MOVL R3,R6 ; Allocated the buffer
0425 1059 JSB G^EXESALONONPAGED ; If LBC not buffer allocated
0425 1060 BLBC R0,20$ ; Retrieve the IRP
0425 1061 MOVL R6,R3 ; Get buffer address in R8
0425 1062 MOVL R2,R8 ; Set that the I/O is from
0425 1063 MOVB #XMTQSM_INTERNAL,- ; an "Internal" IRP
0425 1064 XMTQSB_FLAG(R8)
0425 1065 .IF DF JNX$$$
0425 1066 movl r2,irp$L_fr4(r3)
0425 1067 .ENDC ;DF jnx$$$
0425 1068
0425 1069 MOVL @IRPSL_SVAPTE(R3),R7 ; Get internal IRP buffer addr
0425 1070 BSBW COP_BUFF ; Copy buffer into allcd buff
0425 1071 BSBW COM_XMITFDT ; Branch for common processing
0425 1072 BLBC R0,30$ ; If BC complete in error
0425 1073 POPR #*M<R3,R4,R5,R6,R7,R8,R9> ; Restore registers
0425 1074 RSB
0425 1075
```

38 A3 20D4 8F 3C 0425 1047 MOVZWL #\$\$\$ DEVINACT,IRP\$L_MEDIA(R3) ; Assume device inactive
03 44 A5 E0 0425 1048 BBS #XMSV_STS_ACTIVE,- ; If BS status active
1198 31 0425 1049 UCB\$L_DEVDEPEND(R5),5\$; device is on line
01 E0 0425 1050 BRW IO_DONE ; Else comp request in error
50 2A A3 0425 1051 BBS #IRPSV_FUNC,- ; If BS then receive function
03F8 8F BB 0425 1052 IRPSV_STS(R3),ALT_RCVFDT
59 32 A3 3C 0425 1053 #*M<R3,R4,R5,R6,R7,R8,R9> ; Save registers
2D 13 0425 1054 PUSHB #*M<R3,R4,R5,R6,R7,R8,R9> ; Get the length of the buffer
51 2A 59 C1 0425 1055 MOVZWL IRPSV_BCNT(R3),R9 ; If EQL then bad parameter
56 53 D0 0425 1056 BEQL 10\$; Add the XMTQ length
00000000 GF 16 0425 1057 ADDL3 R9,#XMTQSK_LENGTH,R1 ; Save the IRP
21 50 E9 0425 1058 MOVL R3,R6 ; Allocated the buffer
53 56 D0 0425 1059 JSB G^EXESALONONPAGED ; If LBC not buffer allocated
58 52 D0 0425 1060 BLBC R0,20\$; Retrieve the IRP
80 8F 90 0425 1061 MOVL R6,R3 ; Get buffer address in R8
1F A8 0425 1062 MOVL R2,R8 ; Set that the I/O is from
0425 1063 MOVB #XMTQSM_INTERNAL,- ; an "Internal" IRP
0425 1064 XMTQSB_FLAG(R8)
0425 1065 .IF DF JNX\$\$\$
0425 1066 movl r2,irp\$L_fr4(r3)
0425 1067 .ENDC ;DF jnx\$\$\$
0425 1068
57 2C B3 D0 0425 1069 MOVL @IRPSL_SVAPTE(R3),R7 ; Get internal IRP buffer addr
FFA5 30 0425 1070 BSBW COP_BUFF ; Copy buffer into allcd buff
FF4C 30 0425 1071 BSBW COM_XMITFDT ; Branch for common processing
10 50 E9 0425 1072 BLBC R0,30\$; If BC complete in error
03F8 8F BA 0425 1073 POPR #*M<R3,R4,R5,R6,R7,R8,R9> ; Restore registers
05 0425 1074 RSB
0425 1075

```
38 A3 14 3C 046F 1076 10$: MOVZWL S^#SS$ BADPARAM,IRP$L_MEDIA(R3) ; Set abort status
    03F8 8F BA 0473 1077 20$: POPR #^M<R3,R4,R5,R6,R7,R8,R9> ; Restore registers
    1151 31 0477 1078 BRW IO_DONE ; Complete the request
    047A 1079
    51 58 DO 047A 1080 30$: MOVL R8,R1 ; Save R8 it contains XMTQ buff
    03F8 8F BA 047D 1081 POPR #^M<R3,R4,R5,R6,R7,R8,R9> ; Restore registers
38 A3 50 DO 0481 1082 MOVL R0,IRP$L_MEDIA(R3) ; Set status
    1135 31 0485 1083 BRW TRANSMIT_IO_DONE ; Complete deall the buffer
    0488 1084
    0488 1085 ALT_RCVFDT:
    52 2C A3 DO 0488 1086 MOVL IRP$L_SVAPTE(R3),R2 ; Is there a buffer to reuse
    19 13 048C 1087 BEQL 10$ ; If EQL then no
    2C A3 D4 048E 1088 CLRL IRP$L_SVAPTE(R3) ; Clear so not deallocated
    54 DD 0491 1089 PUSHL R4
50 000020D4 8F DO 0493 1090 MOVL #SS$ DEVINACT,R0 ; Assume failure
54 00A8 C5 DO 049A 1091 MOVL UCB$C_NO_BUFFER(R5),R4 ; Get NOB address
    OD 13 049F 1092 BEQL 20$
    0BD4 30 04A1 1093 BSBW ADDFREELIST ; Else add it to the free list
    54 8ED0 04A4 1094 POPL R4
    003B 30 04A7 1095 10$: BSBW COM_RCVFDT ; Do common processing
    01 50 E9 04AA 1096 BLBC R0,20$ ; Br if unsuccessful
    05 04AD 1097 RSB
    04AE 1098
    38 A3 50 DO 04AE 1099 20$: MOVL R0,IRP$L_MEDIA(R3) ; Set status
    1116 31 04B2 1100 BRW IO_DONE ; Complete IRP in error
    04B5 1101
```



```
04B5 1103 .SBTTL NOSRCVFDT - Receive I/O FDT routine
04B5 1104 :++
04B5 1105 : NOSRCVFDT - Receive I/O FDT routine
04B5 1106 :
04B5 1107 : FUNCTIONAL DESCRIPTION:
04B5 1108 :
04B5 1109 : The specified buffer is checked for accessibility. The buffer address and
04B5 1110 : count are saved in the packet. Then IPL is set to device fork IPL and if
04B5 1111 : a message is available the operation is completed. Otherwise the packet
04B5 1112 : is queued onto the waiting receive list.
04B5 1113 :
04B5 1114 : For requests specifying IOSM_NOW, the I/O is completed with status of
04B5 1115 : $$$_ENDOFIL if no message is available when the test is made.
04B5 1116 :
04B5 1117 :
04B5 1118 : INPUTS:
04B5 1119 : R3 = I/O packet address
04B5 1120 : R4 = PCB address
04B5 1121 : R5 = UCB address
04B5 1122 : R6 = CCB address
04B5 1123 : R7 = Function code
04B5 1124 : AP = Address of first I/O request parameter
04B5 1125 :
04B5 1126 : OUTPUTS:
04B5 1127 : R0 = Status of the receive request
04B5 1128 : R3-R7 preserved.
04B5 1129 :--
04B5 1130 NOSRCVFDT:
04B5 1131 MOVZWL #$$$_BADPARAM,R0 ; Assume illegal size
04B5 1132 MOVZWL P2(AP),R1 ; Get size
04B5 1133 BEQL ABORTIO ; Br if none specified
04B5 1134 MOVL P1(AP),R0 ; Get buffer address
04B5 1135 MOVL R0,IRP$L_MEDIA(R3) ; Save address
04B5 1136 CLRW IRP$W_BOFF(R3) ; No quota to return during
04B5 1137 ; completion
04B5 1138 JSB G^EXE$READCHK ; Check buffer accessibility
04B5 1139 ; (no return on no access)
04B5 1140 SETIPL UCB$B_FIPL(R5) ; Synchronize access to the UCB
04B5 1141 BSBB COM_RCVFDT ; Process the request
04B5 1142 BLBC R0,ABORTIO ; Br if error
04B5 1143 JMP G^EXE$QIORETURN ; Return to await completion
04B5 1144 ABORTIO: ; Abort the I/O request
04B5 1145 CLRL R1 ; Don't return device status
04B5 1146 JMP G^EXE$ABORTIO ;
04B5 1147
04B5 1148
04B5 1149 ; Common receive processing
04B5 1150
04B5 1151 COM_RCVFDT: ; Common receive processing
04B5 1152 BBS #XMSV_STS_ACTIVE,- ; Br if device active
04B5 1153 UCB$L_DEVDEPEND(R5),10$
04B5 1154 MOVZWL #$$$_DEVINACT,R0 ; Set return status
04B5 1155 RSB
04B5 1156
04B5 1157
04B5 1158 ; Check for an available message and complete the receive
04B5 1159
```

50 14 3C 04B5 1131
51 04 AC 3C 04B8 1132
1F 13 04BC 1133
50 6C D0 04BE 1134
38 A3 50 D0 04C1 1135
30 A3 B4 04C5 1136
00000000'GF 16 04C8 1137
04CE 1138
04CE 1139
11 10 04D2 1141
06 50 E9 04D4 1142
00000000'GF 17 04D7 1143
51 D4 04DD 1144
00000000'GF 17 04DF 1146
04E5 1147
04E5 1148
04E5 1149
04E5 1150
04E5 1151
06 44 0B E0 04E5 1152
50 20D4 8F 3C 04E7 1153
05 04EA 1154
04EF 1155
04F0 1156
04F0 1157
04F0 1158
04F0 1159

```
54      54      DD      04F0      1160      10$:      PUSHL      R4
54      00A8      C5      D0      04F2      1161      MOVL      UCB$$_NO_BUFFER(R5),R4      ; Get NOB address
2D      13      04F7      1162      BEQL      30$      ; If eql device not active
52      10      B4      0F      04F9      1163      REMQUE      @NOB$$_ATTN(R4),R2      ; Dequeue a received message
07      1D      04FD      1164      BVS      15$      ; Br if none
1077      30      04FF      1165      BSBW      FINISH_RCV_IO      ; Comp the I/O request
54      8ED0      0502      1166      POPL      R4
05      0505      1167      RSB
0506      1168
0506      1169      ; Queue the request for future message arrival unless IOSM_NOW specified.
0506      1170      ; Receives are queued to the special receive wait queue.
0506      1171
0B      20      A3      06      E0      0506      1172      15$:      BBS      #IOSV_NOW,IRP$W_FUNC(R3),20$      ; Br BS read NOW
1C      B4      63      0E      050B      1173      INSQUE      (R3),@NOB$$_RCVS+4(R4)      ; Queue the I/O packet
54      8ED0      050F      1174      POPL      R4
50      01      3C      0512      1175      MOVZWL      S^#SS$_NORMAL,R0      ; Set QIO status
05      0515      1176      RSB
0516      1177
54      8ED0      0516      1178      20$:      POPL      R4
38      A3      0870      8F      3C      0519      1179      MOVZWL      #SS$_ENDOFFILE,IRP$$_MEDIA(R3)      ; Set no message status
10A9      30      051F      1180      BSBW      IO_DONE      ; Complete the I/O
50      01      3C      0522      1181      MOVZWL      S^#SS$_NORMAL,R0      ; Set normal completion
05      0525      1182      RSB      ; And return
0526      1183
54      8ED0      0526      1184      30$:      POPL      R4
50      20D4      8F      3C      0529      1185      MOVZWL      #SS$_DEVINACT,R0      ; Set error
05      052E      1186      RSB      ; And return
052F      1187
```



```
052F 1189 .SBTTL NOS$SETMODEFDT, Set mode I/O operation FDT routine
052F 1190
052F 1191 :++
052F 1192 : NOS$SETMODEFDT - Set mode I/O operation FDT routine
052F 1193 :
052F 1194 : Functional description:
052F 1195 :
052F 1196 : This routine is used to set the configuration of the terminal device
052F 1197 : including configuration of software DDCMP. The first SETMODE done to the
052F 1198 : device causes a the NOB to be allocated. This buffer is used in place
052F 1199 : of an extension to the devices UCB. Subfunction modifier bits are used
052F 1200 : to specify the type of action to be taken. The two characteristics buffers
052F 1201 : (P1 and P2) are used to describe specific characteristics.
052F 1202 :
052F 1203 : The QIO parameters for SETMODE are:
052F 1204 :
052F 1205 : P1 = Optional address of quadword or longword buffer
052F 1206 : P2 = Optional address of buffer descriptor for extended characteristics
052F 1207 : P3 = Number of receive buffers to pre-allocate. Required on
052F 1208 : controller startup.
052F 1209 :
052F 1210 :
052F 1211 : The subfunction modifiers are as follows:
052F 1212 :
052F 1213 :
052F 1214 : o STARTUP - start the device - this modifier is used to
052F 1215 : start the device.
052F 1216 :
052F 1217 : o SHUTDOWN - shutdown the device - this modifier
052F 1218 : is used to stop the device.
052F 1219 :
052F 1220 : o ATTNAST - request an attention AST - this modifier is used
052F 1221 : to set up an AST to be delivered when a change of
052F 1222 : status occurs on this device.
052F 1223 :
052F 1224 : o CTRL - perform the request on the Controller not the
052F 1225 : tributary.
052F 1226 :
052F 1227 : o SET_MODEM - set line unit's mode register.
052F 1228 :
052F 1229 : INPUTS:
052F 1230 : R3 = IRP address
052F 1231 : R4 = PCB address
052F 1232 : R5 = UCB address
052F 1233 : R6 = CCB address
052F 1234 : R7 = Function code
052F 1235 : AP = address of first QIO parameter
052F 1236 :
052F 1237 : OUTPUTS:
052F 1238 : R0 = status of setmode request
052F 1239 : R3-R5 are preserved.
052F 1240 : R7-R9 = destroyed
052F 1241 :
052F 1242 : --
052F 1243 : NOS$SETMODEFDT:
50 00A8 C5 D0 052F 1244 MOVL UCBSL_NO_BUFFER(R5),R0 ; If NEQ then BUFFER allocated
27 12 0534 1245 BNEQ 3$
```



```
51 000000AA 8F D0 0536 1246      MOVL    #NOB$K_LENGTH,R1      ; Else set up to allocate the buffer
      0265 30 053D 1247      BSBW    ALLOC_BUFFER      ; Call to allocate the buffer
      57 50 E9 0540 1248      BLBC    R0,28$          ; If LBC buffer could not be allocat
      54 DD 0543 1249      PUSHL    R4
      54 52 D0 0545 1250      MOVL    R2,R4          ; Setup NOB address
      027F 30 0548 1251      DSBINT   UCB$B_FIPL(R5)
00A8 C5 54 D0 054F 1252      BSBW    INIT_NO_BUFFER      ; Setup initial states in NOB
      0552 1253      MOVL    R4,UCB$C_NO_BUFFER(R5)    ; Set address of NOB
      0557 1254      ENBINT
      54 8ED0 055A 1255      POPL     R4
      57 20 A3 3C 055D 1256 3$:  MOVZWL  IRP$W_FUNC(R3),R7      ; Get entire function code
      03 57 09 E1 0561 1257      BBC    #IO$V_CTRL,R7,5$      ; Br if not controller request
      00AD 31 0565 1258      BRW     SETMODE_CTRL      ; Process controller request
      0568 1259
      0568 1260      ; Perform setmode request on a tributary
      0568 1261
      34 57 08 E1 0568 1262 5$:  BBC    #IO$V_ATTNA$T,R7,30$      ; Branch if not attention AST
      056C 1263
      056C 1264      ; User is requesting an attention AST.
      056C 1265
      056C 1266
      57 00AC C5 DE 056C 1267      MOVAL    UCB$L_NO_AST(R5),R7      ; Get addr of AST list
      00000000'GF 16 0571 1268      JSB     G^COM$SETATTNA$T      ; Set up attention AST
      50 00A8 C5 D0 0577 1269      MOVL    UCB$L_NO_BUFFER(R5),R0      ; Get NOB address
      51 10 A0 9E 057C 1270      MOVAB    NOB$Q_ATT$N(R0),R1      ; Check for empty rcv list
      61 51 D1 0580 1271      CMPL     R1,(RT)      ; Empty?
      08 13 0583 1272      BEQL     20$      ; Yes, no need to inform user
      53 DD 0585 1273 10$:  PUSHL    R3      ; Else, save IRP address
      1157 30 0587 1274      BSBW    POKE_USER      ; Inform the user
      53 8ED0 058A 1275      POPL     R3      ; Restore IRP address
      51 44 A5 D0 058D 1276 20$:  MOVL    UCB$L_DEVDEPEND(R5),R1      ; Get device characteristics
      50 01 3C 0591 1277 23$:  MOVZWL  S^#SS$NORMAL,R0      ; Set success
      00000000'GF 17 0594 1278 25$:  JMP     G^EXE$FINISHIO      ; Complete the I/O
      00000000'GF 17 059A 1279
      059A 1280 28$:  JMP     G^EXE$ABORTIO      ; Abort the request
      05A0 1281
      05A0 1282
      04DD 30 05A0 1283 30$:  BSBW    GET_CHAR_BUFS      ; Get P1 and P2 characteristics
      30 50 E9 05A3 1284      BLBC    R0,40$      ; Br if error - abort I/O
      32 57 07 E1 05A6 1285      BBC    #IO$V_SHUTDOWN,R7,50$      ; Branch if not trib shutdown
      05AA 1286
      05AA 1287      ; Shutdown tributary modifier specified.
      05AA 1288
      05AA 1289      ; Validate P2 buffer. Then update trib parameter block.
      05AA 1290
      02 90 05AA 1291      MOVAB    S^#NO_FC_V_STOP_CIR,-      ; Set internal function code
      21 A3 05AC 1292      IRP$B_NOFUNC(R3)
      52 44 A5 3C 05AE 1293      MOVZWL  UCB$L_DEVDEPEND(R5),R2      ; Else, get status
      05B2 1294 35$:  DSBINT   UCB$B_FIPL(R5)      ; Sync to get the UCB
      144C 30 05B9 1295      BSBW    VALIDATE_P2_TRIB      ; Validate the P2 buffer
      05BC 1296      ENBINT
      51 44 A5 D0 05BF 1297      MOVL    UCB$L_DEVDEPEND(R5),R1      ; Restore IPL
      CE 50 E9 05C3 1298      BLBC    R0,25$      ; Assume no error
      05C6 1299      DSBINT   UCB$B_FIPL(R5)      ; Br if error
      0617 30 05CD 1300      BSBW    CHG_TRIB      ; Sync to get the UCB
      049A 31 05D0 1301      ENBINT      ; Change trib parameters
      05D3 1302      BRW     QUEPKT      ; Restore IPL
      ; Queue packet to driver
```


00000000'GF	17	05D6	1303	40\$:	JMP	G^EXE\$ABORTIO	: Abort the I/O request
52 44 A5	3C	05D6	1304	50\$:	MOVZWL	UCB\$L_DEVDEPEND(R5),R2	: Get status
141E	30	05DC	1305		DSBINT	UCB\$B_FIPL(R5)	: Sync to get the UCB
E6 50	E9	05E0	1307		BSBW	VALIDATE_P2_TRIB	: Validate the P2 buffer
05ED	30	05E7	1308		ENBINT		: Restore IPL
0D 57 06	E0	05EA	1309		BLBC	R0,40\$	: Br if error
50 01	3C	05ED	1310		DSBINT	UCB\$B_FIPL(R5)	: Sync to get the UCB
51 44 A5	D0	05F0	1311	60\$:	BSBW	CHG_TRIB	: Change trib parameters
00000000'GF	17	05F7	1312		ENBINT		: Restore IPL
00	90	05FA	1313		BBS	#IOSV STARTUP,R7,80\$	: Br if startup request
21 A3	31	05FD	1314		MOVZWL	S^#SS\$ NORMAL,R0	: Else, set successful return
045B	31	0601	1315		MOVL	UCB\$L_DEVDEPEND(R5),R1	: Set IOSB return status
		0604	1316		JMP	G^EXE\$FINISHIO	: Finish the I/O request
		0608	1317	80\$:	MOVB	S^#NO_FC_V_STRT CIR,-	: Set internal function code
		060E	1318			IRP\$B_NOFUNC(R3)	
		0610	1319		BRW	QUEPKT	
		0612	1320				
		0615	1321				


```
0615 1323 .SBTTL SETMODE_CTRL, Perform setmode FDT operation on controller
0615 1324
0615 1325 ;++
0615 1326 ; SETMODE_CTRL - Perform setmode FDT operation on controller
0615 1327 ;
0615 1328 ; Functional description:
0615 1329 ;
0615 1330 ; This routine performs the SETMODE FDT setup for the controller.
0615 1331 ;
0615 1332 ; INPUTS:
0615 1333 ; R3 = IRP address
0615 1334 ; R4 = PCB address
0615 1335 ; R5 = UCB address
0615 1336 ; R7 = IRP function word
0615 1337 ;
0615 1338 ; OUTPUTS:
0615 1339 ; R0 = status of setmode request
0615 1340 ; R3-R5 are preserved.
0615 1341 ;
0615 1342 ;--
0615 1343 SETMODE_CTRL:
0615 1344 BSBW GET_CHAR_BUFS ; Perform setmode on controller
0618 1345 BLBC R0,20$ ; Get P1 and P2 characteristics
061B 1346 ; Br if error - Abort I/O
061B 1347 BITW #<XMSM CHR CTRL!- ; If NEQ then not multipoint
061C 1348 XMSM CHR DMC>,- ; control or DMC mode
061C 1349 IRP$C_MEDIA+4(R3)
0621 1350 BEQL 5$ ;
0623 1351 MOVZWL #SS$_BADPARAM,R0 ; Else set bad parameter
0626 1352 BRB 20$
0628 1353 5$: CLRL R6 ; Assume no protocol specified
062A 1354 MOVZWL #NMASC_PCLI_PRO,R1 ; Check P2 buffer for multi
062F 1355 DSBINT UCBSB_FIPL(R5) ; Sync to get the UCB
0636 1356 BSBW UNPACK_P2_BUF ; point control specification
0639 1357 ENBINT ; Restore IPL
063C 1358 BLBC R0,14$ ; If BC then no prot specified
063F 1359 CMPB #NMASC_LINPR_CON,R2 ; If EQL then multipoint contrl
0642 1360 BEQL 10$ ; specified
0644 1361 BRB 14$
0646 1362 10$: MOVZWL #SS$_BADPARAM,R0 ; Else set bad parameter
0649 1363 BRB 17$
064B 1364 14$: MOVZWL UCBSW_DEVSTS(R5),R2 ; Get device status
064F 1365 DSBINT UCBSB_FIPL(R5) ; Sync to get the UCB
0656 1366 BSBW VALIDATE_P2_UCB ; Validate the P2 buffer
0659 1367 ENBINT ; Restore IPL
065C 1368 BLBC R0,20$ ; Br if error
065F 1369 BBC #IO$V_SHUTDOWN,R7,30$ ; Br if not shutdown request
0663 1370 ;
0663 1371 ; Shutdown modifier specified
0663 1372 ;
0663 1373 MOVB S^#NO_FC V STOP LIN,- ; Set internal function code
0665 1374 IRP$B_NOFUNC(R3)
0667 1375 DSBINT UCBSB_FIPL(R5) ; Sync to get NOB
066E 1376 BSBW CHG_UCB_NOB ; Update the UCB and the NOB
0671 1377 ENBINT ; Lower IPL
0674 1378 BBC #NO_DS V INITED,- ; Br if controller not up
0676 1379 UCBSW_DEVSTS(R5),15$
```



```
03F4 31 0679 1380 BRW QUEPKT ; Queue packet to driver
      067C 1381
51 44 A5 D0 067C 1382 15$: MOVL UCB$ DEVDEPEND(R5),R1 ; Get IOSB1 return
50 01 3C 0680 1383 ; MOVZWL S^#SS$ NORMAL,R0 ; Set success
00000000'GF 17 0683 1384 17$: JMP G^EXE$FINISHIO ; Complete the I/O request
      0689 1385
00000000'GF 17 0689 1386 20$: JMP G^EXE$ABORTIO ; Abort the I/O request
      068F 1387
00C2 31 068F 1388 25$: BRW 50$ ; Branch to compl request
      0692 1389
F9 57 06 E1 0692 1390 30$: BBC #IOSV_STARTUP,R7,25$ ; Br if not startup request
      01 90 0696 1391 ; MOVB S^#NO_FC V STRT LIN,- ; Set internal function code
      21 A3 0698 1392 ; IRP$B_NOFUNC(R3)
51 08 AC 3C 069A 1393 ; MOVZWL P3(APT,R1) ; If EQL then P3 not set up
      28 13 069E 1394 ; BEQL 35$
50 00A8 C5 D0 06A0 1395 ; MOVL UCB$ NO_BUFFER(R5),R0 ; Get NOB address
      06 E1 06A5 1396 ; BBC #NO_DS V INITED,- ; If BC then device not inited
      10 68 A5 06A7 1397 ; UCB$W_DEVSTS(R5),33$ ; set number of buffers
73 A0 51 91 06AA 1398 ; CMPB R1,NOB$B_BFN(R0) ; Are the buffer nmb's the same
      18 13 06AE 1399 ; BEQL 35$ ; Br if yes
51 0451 8F 3C 06B0 1400 ; MOVZWL #NMASC PCLI BFN,R1 ; Set IOSB1 return
50 14 3C 06B5 1401 ; MOVZWL #SS$_BADPARAM,R0 ; Set error return
      C9 11 06B8 1402 ; BRB 17$ ; Finish the I/O request
73 A0 51 90 06BA 1403 33$: DSBINT UCB$B FIPL(R5) ; Sync to get NOB
      06C1 1404 ; MOVB R1,NOB$B_BFN(R0) ; Store new rcve buffer number
      06C5 1405 ; ENBINT ; Restore IPL
      06C8 1406
      06C8 1407 ; Set new P1, P2 parameters
      06C8 1408
03 68 A5 E1 06C8 1409 35$: BBC #NO_DS V INITED,- ; Br if device not already
00AA 31 06CA 1410 ; UCB$W_DEVSTS(R5),36$ ; inited
      06D0 1411 ; BRW 70$
03 38 A3 E9 06D7 1412 36$: DSBINT UCB$B FIPL(R5) ; Sync to get UCB
44 A5 94 06DB 1413 ; BLBC IRP$B_MEDIA(R3),37$ ; Br if no P1
049B 30 06DE 1414 ; CLRB UCB$ DEVDEPEND(R5) ; Clear old characteristics
      06E1 1415 37$: BSBW CHG_UCB_NOB ; Update the UCB and the NOB
50 00A8 C5 D0 06E4 1416 ; ENBINT ; Restore IPL
51 73 A0 9A 06E9 1417 ; MOVL UCB$ NO_BUFFER(R5),R0 ; Get NOB address
52 3A A3 3C 06ED 1418 ; MOVZBL NOB$B_BFN(R0),R1 ; Get number of receive buffers
04 38 A3 E8 06F1 1419 ; MOVZWL IRP$B_MEDIA+2(R3),R2 ; Get message size from P1
52 42 A5 3C 06F5 1420 ; BLBS IRP$B_MEDIA(R3),40$ ; Br if P1 buffer is valid
50 14 3C 06F9 1421 ; MOVZWL UCB$W_DEVBUFSIZ(R5),R2 ; Else, get buff size from UCB
52 51 C4 06FC 1422 40$: MOVZWL #SS$_BADPARAM,R0 ; Assume bad parameter
      73 13 06FF 1423 ; MULL R1,R2 ; Compute total needed for buff
57 52 3C 0701 1424 ; BEQL 60$ ; Br if zero - error
57 52 D1 0704 1425 ; MOVZWL R2,R7 ; Copy quota
      6B 12 0707 1426 ; CMPL R2,R7 ; Overflow?
53 DD 0709 1427 ; BNEQ 60$ ; Br if error
00000000'GF 16 070B 1428 ; PUSHL R3 ; Save R3
53 8ED0 0711 1430 ; JSB G^EXE$BUFQUOPRC ; Check caller's quota
50 50 E9 0714 1431 ; POPL R3 ; Restore R3
40 A3 57 B0 0717 1432 ; BLBC R0,60$ ; Br if error
50 0080 C4 D0 071B 1433 ; MOVW R7,IRP$W_QUOTA(R3) ; Save quota in packet
20 A0 57 C2 0720 1434 ; MOVL PCB$ JIB(R4),R0 ; Get JIB address
      0724 1435 ; SUBL R7,JIB$B_BYTCNT(R0) ; Charge user for rec. bufs
      0724 1436 ; The following call is used to allocate a buffer for the protocol, DDCMP.
```


				0724	1437	:	If the routine returns with a LBC the the buffer could not be allocated and
				0724	1438	:	the device can not be started.
				0724	1439		
51	2C	000001C8	8F	C1	0724	1440	ADDL3 #TFSK_LENGTH,#GFSK_LENGTH,R1
			0076	30	072C	1441	BSBW ALLOC_BUFFER
			42 50	E9	072F	1442	BLBC R0,60\$; If LBC then abort startup
					0732	1443	DSBINT UCB\$B_FIPL(R5)
	50	00A8	C5	D0	0739	1444	MOVL UCB\$L_NO_BUFFER(R5),R0 ; Get NOB address
		0C A0	52	D0	073E	1445	MOVL R2,NOB\$A_PRO_BUFFER(R0) ; Set protocol address
			3F	BB	0742	1446	PUSHR #^M<R0,RT,R2,R3,R4,R5> ; Before queueing request zero
01C8	C2	2C	00 65	2C	0744	1447	MOVCS #0,(R5),#0,#GFSK_LENGTH,TFSK_LENGTH(R2); the GFB portion of
			3F	BA	074C	1448	POPR #^M<R0,R1,R2,R3,R4,R5> ; buffer
					074E	1449	ENBINT
		031C	31	0751	1450		BRW QUEPKT ; Queue request to driver
				0754	1451		
				0754	1452	:	No modifier specified - change controller parameters
				0754	1453		
			06	E0	0754	1454	50\$: SETIPL UCB\$B_FIPL(R5) ; Sync access to UCB
					0758	1455	BBS #NO_DS_V_INITED,- ; Br if already initd
	1D	68	A5		075A	1456	UCB\$W_DEVSTS(R5),70\$
	03	38	A3	E9	075D	1457	BLBC IRP\$L_MEDIA(R3),53\$; Br if no P1 buffer
		44	A5	94	0761	1458	CLRB UCB\$L_DEVDEPEND(R5) ; Clear old UCB characteristics
			0415	30	0764	1459	53\$: BSBW CHG_UCB_NOB ; Update the UCB and the NOB
		50	01	3C	0767	1460	MOVZWL S^#SS\$ NORMAL,R0 ; Else, set success
	51	44	A5	D0	076A	1461	MOVL UCB\$L_DEVDEPEND(R5),R1 ; Set IOSB1 return
		00000000	'GF	17	076E	1462	55\$: JMP G^EXE\$FINISHIO ; Finish the I/O request
					0774	1463	
		00000000	'GF	17	0774	1464	60\$: JMP G^EXE\$ABORTIO ; Abort the I/O request
					077A	1465	
					077A	1466	; Device already initd - set new parameters and give them to device
					077A	1467	
	OF	38	A3	E9	077A	1468	70\$: BLBC IRP\$L_MEDIA(R3),80\$; Br if no P1 buffer
		3C	A3	91	077E	1469	CMPB IRP\$L_MEDIA+4(R3)- ; Are characteristics okay?
		44	A5		0781	1470	UCB\$L_DEVDEPEND(R5)
			08	13	0783	1471	BEQL 80\$; Yes - let it go
		50	14	3C	0785	1472	MOVZWL #SS\$ BADPARAM,R0 ; Return error
	51		01	CE	0788	1473	MNEGL S^#1,R1 ; No specific parameter
			E1	11	078B	1474	BRB 55\$; Complete the I/O
					078D	1475	
					078D	1476	80\$: SETIPL UCB\$B_FIPL(R5) ; Sync to get UCB
		03E8	30	0791	1477		BSBW CHG_UCB_NOB ; Update the UCB and the NOB
	50	F0	8F	90	0794	1478	MOVW #<15a4>,R0 ; Set only four parameters
		50	01	B0	0798	1479	MOVW S^#SS\$ NORMAL,R0
	51	44	A5	D0	079B	1480	MOVL UCB\$L_DEVDEPEND(R5),R1
		00000000	'GF	17	079F	1481	JMP G^EXE\$FINISHIO


```
07A5 1483 .SBTTL ALLOC_BUFFER, Allocate a buffer
07A5 1484 :++
07A5 1485 :ALLOC_BUFFER - Allocate protocol buffer routine
07A5 1486 :
07A5 1487 : This routine does all the necessary operations to allocate a buffer from
07A5 1488 : nonpaged pool.
07A5 1489 :
07A5 1490 : INPUTS:
07A5 1491 : R1 = Buffer size
07A5 1492 : R4 = Address of the PCB
07A5 1493 :
07A5 1494 : OUTPUTS:
07A5 1495 : R2 = Address of buffer allocated
07A5 1496 : R3 is preserved
07A5 1497 :
07A5 1498 :--
07A5 1499 ALLOC_BUFFER:
07A5 1500 PUSH R3
07A7 1501 JSB G^EXE$BUFFRQUOTA ; Does the user have quota
07AD 1502 BLBC R0,10$ ; If LBC no quota
07B0 1503 JSB G^EXE$ALLOCBUF ; Allocate the buffer
07B6 1504 BLBC R0,10$ ; If LBC then not allocated
07B9 1505 MOVL PCB$L_JIB(R4),R0 ; Get the users JIB
07BE 1506 SUBL2 R1,JIB$L_BYTCNT(R0) ; Subtract the quota
07C2 1507 MOVW R1,UCB$W_SIZE(R2) ; Save the size of BUFFER allocated
07C6 1508 MOVW S^#DYN$C_BUFIO,UCB$B_TYPE(R2) ; Set buffer type
07CA 1509 MOVZWL S^#SS$_NORMAL,R0 ; Set successful return
07CD 1510 10$: POPL R3
07D0 1511 RSB
07D1 1512
```

00000000'GF 53 DD 07A5 1500
1D 50 E9 07A7 1501
00000000'GF 16 07AD 1502
14 50 E9 07B0 1503
50 0080 C4 D0 07B6 1504
20 A0 51 C2 07B9 1505
08 A2 51 B0 07BE 1506
0A A2 13 B0 07C2 1507
50 01 3C 07C6 1508
53 8ED0 07CA 1509
05 07CD 1510
07D0 1511
07D1 1512

```
07D1 1514 .SBTTL INIT_NO_BUFFER - Initialize NOB extension
07D1 1515
07D1 1516 :++
07D1 1517 : INIT_NO_BUFFER - Initialize NOB extension
07D1 1518 :
07D1 1519 : Functional description:
07D1 1520 :
07D1 1521 : This routine is called to initialize queues and reset parameters when the
07D1 1522 : NOB has been allocated on a startup.
07D1 1523 :
07D1 1524 : INPUTS:
07D1 1525 : R4 = NOB address
07D1 1526 : R5 = UCB address
07D1 1527 :
07D1 1528 : IPL = FIPL
07D1 1529 :
07D1 1530 : OUTPUTS:
07D1 1531 : R0-R2 are destroyed.
07D1 1532 :
07D1 1533 :--
07D1 1534 INIT_NO_BUFFER:
07D1 1535
07D1 1536 ; Initialize queue headers
07D1 1537
18 A4 18 A4 9E 07D1 1538 MOVAB NOB$Q_RCVS(R4),NOB$Q_RCVS(R4) ; Receive list
1C A4 18 A4 9E 07D6 1539 MOVAB NOB$Q_RCVS(R4),NOB$Q_RCVS+4(R4)
20 A4 20 A4 9E 07DB 1540 MOVAB NOB$Q_FREE(R4),NOB$Q_FREE(R4) ; Free buffer list
24 A4 20 A4 9E 07E0 1541 MOVAB NOB$Q_FREE(R4),NOB$Q_FREE+4(R4)
28 A4 28 A4 9E 07E5 1542 MOVAB NOB$Q_POST(R4),NOB$Q_POST(R4) ; Post list
2C A4 28 A4 9E 07EA 1543 MOVAB NOB$Q_POST(R4),NOB$Q_POST+4(R4)
10 A4 10 A4 9E 07EF 1544 MOVAB NOB$Q_ATTN(R4),NOB$Q_ATTN(R4) ; Full buffer list
14 A4 10 A4 9E 07F4 1545 MOVAB NOB$Q_ATTN(R4),NOB$Q_ATTN+4(R4)
07F9 1546
07F9 1547 ASSUME NOB$L_XMT_INPR EQ NOB$L_RCV_INPR+4
07F9 1548
30 A4 7C 07F9 1549 CLRQ NOB$L_RCV_INPR(R4)
42 A4 94 07FC 1550 CLRB NOB$B_XSTATE(R4) ; Clear xmt state
44 A4 FFFF 8F B0 07FF 1551 MOVW #NOB$C_PAD,NOB$W_PADS(R4) ; Set up field with pads to send
0805 1552
0805 1553
0805 1554 ; Enable the receiver. DDCMP STRT messages could be coming in on the
0805 1555 ; wire and we will start looking at them as soon as this buffer is
0805 1556 ; given to the device. ( The address is stored in UCB$L_NO_BUFFER.)
0805 1557 ; Since that is the case then we might as well start framing receive messages
0805 1558 ; as well. The protocol will not be brought up however until both
0805 1559 ; the line and circuit are turned on.
0805 1560
0805 1561 DSBINT UCB$B_DIPL(R5)
08BE 30 080C 1562 BSBW START_RECEIVE
080F 1563 ENBINT
0812 1564
0812 1565
0812 1566 ; Set up the block allocated in the NOB to look like a DDCMP control message
0812 1567 ; buffer
0812 1568
50 0080 C4 DE 0812 1569 MOVAL NOB$Z_CTL_MSG(R4),R0
OC A0 D4 0817 1570 CLRL XMTQ$C_IRP(R0)
```



```
1F A0 04 90 081A 1571      MOVB    #XMTQSM_CONTROL,XMTQSB_FLAG(R0)
              081E 1572
              081E 1573 ; Please note that setting this of this parameter on device startup is
              081E 1574 ; accomplished via the NOBSW_DEVBUSIZ field in the NOB buffer. This
              081E 1575 ; was done to make the validating and the setting of this field simpler
              081E 1576
42 A5 0100 8F B0 081E 1577      MOVW    #NOSC_DEF_BUFSIZ,UCBSW_DEVBUSIZ(R5) ; Set default buffer size
              50 06 3C 0824 1578      MOVZWL  #DEF_LINE_PARAMSZ,R0 ; Set size of defaults in bytes
51 F8D3 CF 9E 0827 1579      MOVAB  DEF_LINE_PARAM,R1 ; Set address of defaults
52 70 A4 9E 082C 1580      MOVAB  NOB$C_SETPRM(R4),R2 ; Set address of parameters
              82 81 90 0830 1581 10$:  MOVB    (R1)+(R2)+ ; Set next default
              FA 50 F5 0833 1582      SOBGTR  R0,10$ ; Loop on all parameters
              50 0A 3C 0836 1583      MOVZWL  #DEF_TRIB_PARAMSZ,R0 ; Set size of defaults in bytes
51 F8B7 CF 9E 0839 1584      MOVAB  DEF_TRIB_PARAM,R1 ; Set address of defaults
52 60 A4 9E 083E 1585      MOVAB  NOB$Z_DDMP(R4),R2 ; Set address of parameters
              82 81 90 0842 1586 20$:  MOVB    (R1)+(R2)+ ; Set next default
              FA 50 F5 0845 1587      SOBGTR  R0,20$ ; Loop on all parameters
              05 0848 1588      RSB
              0849 1589
```

```
0849 1591 .SBTTL NOSSENSEMODEFDT, Sense Mode I/O operation FDT routine
0849 1592
0849 1593 :++
0849 1594 : NOSSENSEMODEFDT - Sense Mode FDT routine
0849 1595
0849 1596 : Functional Description:
0849 1597 :
0849 1598 : This routine returns information to the caller about the configuration
0849 1599 : and status of the DMF32 device. Depending on the function modifier,
0849 1600 : either the device characteristics, error counters contents are returned.
0849 1601 :
0849 1602 :
0849 1603 : The QIO parameters for SENSEMODE are:
0849 1604 :
0849 1605 : P1 = optional address of quadword or longword buffer
0849 1606 : P2 = optional address of buffer descriptor for extended characteristics
0849 1607 :
0849 1608 :
0849 1609 : INPUTS:
0849 1610 : R3 = IRP address
0849 1611 : R4 = PCB address
0849 1612 : R5 = UCB address
0849 1613 : R6 = CCB address
0849 1614 : R7 = Function code
0849 1615 : AP = Address of first function-dependent QIO parameter
0849 1616 :
0849 1617 : OUTPUTS:
0849 1618 : R0 = status return of sensemode request
0849 1619 : R3,R5 are preserved.
0849 1620 :
0849 1621 :--
0849 1622 NOSSENSEMODEFDT:
0849 1623 BBS #NO_DS_V_INITED,UCB$W_DEVSTS(R5),2$ ; If inited then ok to read
0849 1624 ; errors and characteristics
0849 1625 MOVZWL #SS$ NORMAL,R0 ; Else return no information
0849 1626 MOVL UCB$C_DEVDEPEND(R5),R1
0849 1627 JMP G^EXE$FINISHIO
0849 1628
0849 1629 2$: MOVW IRP$W_FUNC(R3),R7 ; Get entire function code
0849 1630
0849 1631 BBC #IOSV_CTRL,R7,5$ ; Br if not controller request
0849 1632 BRW SENSEMODE_CTRL ; Else, process controller req.
0849 1633 5$: BITW #<IOSM_RD_COUNT!- ; Check to see if either bit is
0849 1634 IOSM_CCR_COUNT>,R7 ; Is set
0849 1635 BEQL 40$ ; If EQL then read parameters
0849 1636 CLRL R0 ; Assume clear count
0849 1637 MOVZWL S^#SS$ NORMAL,R1 ; Assume success
0849 1638 BBC #IOSV_RD_COUNT,R7,15$ ; If BC then not read count
0849 1639
0849 1640 ; Read tributary counters - modifier RD_COUNT
0849 1641
0849 1642 BSBW CHECK P2 ; Check P2 buffer
0849 1643 MOVZWL #SS$ BADPARAM,R0 ; Assume zero length buffer
0849 1644 MOVW R1,IRP$L_MEDIA+4(R3) ; Save user size of P2 buffer
0849 1645 BEQL 30$ ; Br if none
0849 1646 PUSHL R4
0849 1647 MOVZWL S^#SS$ NORMAL,R1 ; Assume success
```

OD 68 A5 06 E0 0849 1623 BBS #NO_DS_V_INITED,UCB\$W_DEVSTS(R5),2\$; If inited then ok to read
50 01 3C 0849 1624 ; errors and characteristics
51 44 A5 D0 0849 1625 MOVZWL #SS\$ NORMAL,R0 ; Else return no information
00000000'GF 17 0851 1626 MOVL UCB\$C_DEVDEPEND(R5),R1
57 20 A3 B0 0855 1627 JMP G^EXE\$FINISHIO
03 57 09 E1 085B 1628
57 0500 8F B3 085B 1629 2\$: MOVW IRP\$W_FUNC(R3),R7 ; Get entire function code
7C 13 085F 1630
51 01 3C 085F 1631 BBC #IOSV_CTRL,R7,5\$; Br if not controller request
3F 57 08 E1 0863 1632 BRW SENSEMODE_CTRL ; Else, process controller req.
026E 30 0866 1633 5\$: BITW #<IOSM_RD_COUNT!- ; Check to see if either bit is
50 14 3C 086B 1634 IOSM_CCR_COUNT>,R7 ; Is set
3C A3 51 B0 086B 1635 BEQL 40\$; If EQL then read parameters
5D 13 086D 1636 CLRL R0 ; Assume clear count
51 01 3C 086F 1637 MOVZWL S^#SS\$ NORMAL,R1 ; Assume success
51 01 3C 0872 1638 BBC #IOSV_RD_COUNT,R7,15\$; If BC then not read count
0876 1639
0876 1640 ; Read tributary counters - modifier RD_COUNT
0876 1641
0876 1642 BSBW CHECK P2 ; Check P2 buffer
50 14 3C 0879 1643 MOVZWL #SS\$ BADPARAM,R0 ; Assume zero length buffer
3C A3 51 B0 087C 1644 MOVW R1,IRP\$L_MEDIA+4(R3) ; Save user size of P2 buffer
5D 13 0880 1645 BEQL 30\$; Br if none
51 01 3C 0882 1646 PUSHL R4
51 01 3C 0884 1647 MOVZWL S^#SS\$ NORMAL,R1 ; Assume success


```
54 00A8 C5 D0 0887 1648      MOVL      UCB$$_NO_BUFFER(R5),R4      ; Get NOB address
      57 13 088C 1649      BEQL      35$      ; If eql no info
54 0C A4 D0 088E 1650      MOVL      NOB$$_PRO_BUFFER(R4),R4      ; Get prot buffer address
      51 13 0892 1651      BEQL      35$      ; If eql no info
50 1E A4 3C 0894 1652      MOVZWL     TFSW_TEG(R4),R0      ; Get size of buffer needed
50 3C A3 B1 0898 1653      CMPW      IRP$$_MEDIA+4(R3),R0      ; IF GEQU then buffer ok
      09 1E 089C 1654      BGEQU     10$      ;
51 0601 8F 3C 089E 1655      MOVZWL     #SS$$_BUFFEROVF,R1      ; Return partial success
50 3C A3 3C 08A3 1656      MOVZWL     IRP$$_MEDIA+4(R3),R0      ; Set size of copy
      3B BB 08A7 1657 10$:  PUSHR      #^M<R0,R1,R3,R4,R5>      ; Save the registers
62 018A C4 50 28 08A9 1658      MOV3     R0,TFSK_ERRSTRT(R4),(R2)      ; Get the errors
      3B BA 08AF 1659      POPR      #^M<R0,R1,R3,R4,R5>      ;
16 57 0A E1 08B1 1660      BBC       #IOSV CLR COUNT,R7,20$      ; Br if not clear counts
      57 03 3C 08B5 1661 15$:  MOVZWL     #<DLK$$_TRIB!DLK$$_CLEAR>,R7      ; Set counters to read/clear
      2B BB 08B8 1662      SETIPL     UCB$$_FIPL(R5)      ; Set to fork IPL
      58 D4 08BC 1663      PUSHR      #^M<R0,R1,R3,R5>      ; Save registers
      55 54 D0 08C0 1664      CLRL      R8      ; Set no parameters to return
      56 03 9A 08C3 1665      MOVL      R4,R5      ; Set address of trib block
      F737' 30 08C6 1666      MOVZBL     #DLK$$_REQEBA,R6      ; Set operation to perform
      2B BA 08C9 1667      BSBW      DDCMP      ; Branch to protocol
50 50 10 78 08CB 1669 20$:  POPR      #^M<R0,R1,R3,R5>      ; Shift size of buffer return
      50 51 B0 08CF 1670      ASHL      #16,R0,R0      ; Set success status
      54 8ED0 08D2 1671      MOVW      R1,R0
51 44 A5 D0 08D5 1672      POPL      R4
00000000'GF 17 08D9 1673      MOVL      UCB$$_DEVDEPEND(R5),R1      ; Set devdepend info
      50 D4 08E5 1676 30$:  JMP       G^EXE$FINISHIO      ; Complete the request
      E2 11 08E7 1678      JMP       G^EXE$ABORTIO      ; Abort the I/O request
      08E9 1679      CLRL      R0
      08E9 1680 35$:  BRB       20$
      08E9 1681      ; Read tributary parameters
50 01 3C 08E9 1682 40$:  MOVZWL     S^#SS$$_NORMAL,R0      ; Assume success
51 08 3C 08EC 1683      MOVZWL     S^#8,RT      ; Size of P1 buffer if present
      01F3 30 08EF 1684      BSBW      CHECK_BUFS      ; Check P1 and P2 buffers
3C A3 51 B0 08F2 1685      MOVW      R1,IRP$$_MEDIA+4(R3)      ; Save user P2 buffer length
      45 13 08F6 1686      BEQL      60$      ; Br if no P2 buffer present
      08F8 1687
3E A3 01 B0 08F8 1688      MOVW      S^#SS$$_NORMAL,IRP$$_MEDIA+6(R3)      ; Assume success
32 A3 18 B0 08FC 1689      MOVW      #TRIB_PRM_BUFSIZ,IRP$$_BCNT(R3)      ; Set size of required buffer
      18 51 B1 0900 1690      CMPW      R1,#TRIB_PRM_BUFSIZ      ; If GEQU then user buffer is
      0A 1E 0903 1691      BGEQU     50$      ; large enough
3E A3 0601 8F B0 0905 1692      MOVW      #SS$$_BUFFEROVF,IRP$$_MEDIA+6(R3)      ; Return partial success
32 A3 51 B0 090B 1693      MOVW      R1,IRP$$_BCNT(R3)      ; Set size to use in move
51 F7C1 CF 9E 090F 1694 50$:  MOVAB     TRIB_PARAM,R1      ; Get address of return table
      54 DD 0914 1695      PUSHL     R4
54 00A8 C5 D0 0916 1696      MOVL      UCB$$_NO_BUFFER(R5),R4      ; Get NOB address
      06 13 091B 1697      BEQL      55$      ; If Neql buffer is available
54 60 A4 9E 091D 1698      MOVAB     NOB$$_DDCMP(R4),R4      ; Get trib param block address
      08 12 0921 1699      BNEQ      56$      ; If Neql buffer is available
00000032'EF B4 0923 1700 55$:  CLRW      IRP$$_BCNT
      03 11 0929 1701      BRB       58$
      1211 30 092B 1702 56$:  BSBW      RETURN_P2      ; Return the P2 parameters
      54 8ED0 092E 1703 58$:  POPL      R4
50 32 A3 B0 0931 1704      MOVW      IRP$$_BCNT(R3),R0      ; Return size of buffer
```


NODRIVER
V04-001

L 15
- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19
NOSSENSEMODEFDT, Sense Mode I/O operati 6-SEP-1984 16:22:41

VAX/VMS Macro V04-00 Page 40
[DRIVER.SRC]NODRIVER.MAR;2 (28)

50	50	10	78	0935	1705	ASHL	#16,R0,R0	; Shift size of buffer return		
	50	3E	A3	B0	0939	1706	MOVW	IRP\$L_MEDIA+6(R3),R0	; Set size of return	
					093D	1707				
	52	38	A3	D0	093D	1708	60\$:	MOVL	IRP\$L_MEDIA(R3),R2	; Retrieve P1 buffer address
			09	13	0941	1709		BEQL	70\$	; Br if none
	62	40	A5	7D	0943	1710		MOVQ	UCB\$B_DEVCLASS(R5),(R2)	; Else, return characteristics
04	A2	44	A5	C8	0947	1711		BISL	UCB\$L_DEVDEPEND(R5),4(R2)	
	51	44	A5	D0	094C	1712	70\$:	MOVL	UCB\$L_DEVDEPEND(R5),R1	; Get device dependend info
	00000000	'GF	17	0950	1713			JMP	G^EXE\$FINISHIO	; Complete the I/O request


```
0956 1715 .SBTTL SENSEMODE_CTRL, Perform SENSEMODE FDT processing for controller
0956 1716
0956 1717 :++
0956 1718 : SENSEMODE_CTRL - Perform SENSEMODE FDT processing for controller
0956 1719 :
0956 1720 : Functional description:
0956 1721 :
0956 1722 : This routine performs all FDT checking for a controller SENSEMODE request.
0956 1723 : The P2 buffer if present is check for write access and if okay, a system
0956 1724 : buffer is allocated for temporarily saving the needed information. The
0956 1725 : P1 sensemode information is returned through IRP$L_MEDIA and IRP$L_MEDIA+4.
0956 1726 :
0956 1727 : INPUTS:
0956 1728 : R3 = IRP address
0956 1729 : R4 = PCB address
0956 1730 : R5 = UCB address
0956 1731 : R7 = Function code
0956 1732 :
0956 1733 : OUTPUTS:
0956 1734 : R0 = status return for request
0956 1735 : R3-R5 are preserved.
0956 1736 :
0956 1737 :--
0956 1738 SENSEMODE_CTRL:
57 0500 8F B3 0956 1739 BITW #<IOSM_RD_COUNT!- ; Process controller sensemode FDT
0956 1740 IOSM_CLR_COUNT>,R7 ; Check to see if either bit is
0956 1741 BNEQ 5$ ; Is set
0956 1742 BRW 40$ ; If NEQ then not read parameter
03 57 0092 31 095D 1743 5$: BBC #IOSV_RD_MODEM,R7,8$ ; else read parameters
00EA 31 0964 1744 8$: BRW SENSE_MODEM ; If BC not a read modem IO
50 D4 0967 1745 8$: CLRL R0 ; Else branch to read modem FDT
51 01 3C 0969 1746 MOVZWL S^#SS$ NORMAL,R1 ; Assume clear count
52 00A8 C5 D0 096C 1747 MOVL UCB$L_NO_BUFFER(R5),R2 ; Assume success
7B 13 0971 1748 BEQL 35$ ; Get NOB address
OC A2 D5 0973 1749 TSTL NOB$A_PRO_BUFFER(R2) ; If eql no buffer available
76 13 0976 1750 BEQL 35$ ; Check for address of protocol buff
3F 57 08 E1 0978 1751 BBC #IOSV_RD_COUNT,R7,15$ ; If eql no buffer available
097C 1752 ; Br if not read counters
097C 1753 ; Read controller counters - modifier RD_COUNT for DDCMP mode only
097C 1754
097C 1755 BSBW CHECK P2 ; Check P2 buffer
50 0168 30 097F 1756 MOVZWL #SS$ BADPARAM,R0 ; Assume zero length buffer
3C A3 51 B0 0982 1757 MOVW R1,IRP$L_MEDIA+4(R3) ; Save user size of P2 buffer
60 13 0986 1758 BEQL 30$ ; Br if none
54 DD 0988 1759 PUSHL R4
54 00A8 C5 D0 098A 1760 MOVL UCB$L_NO_BUFFER(R5),R4 ; Get NOB address
54 OC A4 D0 098F 1761 MOVL NOB$A_PRO_BUFFER(R4),R4 ; Set address of protocol buffer
51 01 3C 0993 1762 MOVZWL S^#SS$ NORMAL,R1 ; Assume success
50 01D3 C4 3C 0996 1763 MOVZWL GF$W_GEB+TF$K_LENGTH(R4),R0 ; Get size of buffer needed
50 3C A3 B1 099B 1764 CMPW IRP$C_MEDIA+4(R3),R0 ; IF GEQU then buffer ok
09 1E 099F 1765 BGEQU 10$
51 0601 8F 3C 09A1 1766 MOVZWL #SS$ BUFFEROVF,R1 ; Return partial success
50 3C A3 3C 09A6 1767 MOVZWL IRP$C_MEDIA+4(R3),R0 ; Set size of copy
3B BB 09AA 1768 10$: PUSHR #^M<R0,R1,R3,R4,R5> ; Save the registers
62 01E7 C4 50 28 09AC 1769 MOV C3 R0,GF$K_ERRSRT+TF$K_LENGTH(R4),(R2) ; Get the errors
3B BA 09B2 1770 POPR #^M<R0,R1,R3,R4,R5>
54 8ED0 09B4 1771 POPL R4
```



```
1C 57 0A E1 09B7 1772 BBC #IOSV CLR COUNT,R7,20$ ; Br if not clear counts
57 05 3C 09BB 1773 15$: MOVZWL #<DLKSM_GLOB!DLKSM_CLEAR>,R7 ; Set counters to read/clear
09BE 1774 SETIPL UCBSB FIPL(R5) ; Set to fork IPL
3B BB 09C2 1775 PUSHF #^M<R0,R1,R3,R4,R5> ; Save registers
58 D4 09C4 1776 CLRL R8 ; Set no parameters to return
55 00A8 C5 D0 09C6 1777 MOVL UCBSL_NO_BUFFER(R5),R5 ; Get NOB address
55 0C A5 D0 09CB 1778 MOVL NOBSA_PRO_BUFFER(R5),R5 ; Set address of protocol buffer
56 03 9A 09CF 1779 MOVZBL #DLKSC_REQEBA,R6 ; Set operation to perform
F62B' 30 09D2 1780 BSBW DDCMP ; Branch to protocol
3B BA 09D5 1781 POPR #^M<R0,R1,R3,R4,R5>
50 50 10 78 09D7 1782 20$: ASHL #16,R0,R0 ; Shift size of buffer return
50 50 51 B0 09DB 1783 MOVW R1,R0 ; Set success status
51 44 A5 D0 09DE 1784 MOVL UCBSL_DEVDEPEND(R5),R1 ; Set devdepend info
00000000'GF 17 09E2 1785 JMP G^EXE$FINISHIO ; Complete the request
00000000'GF 17 09E8 1786 30$: JMP G^EXE$ABORTIO ; Abort the I/O request
50 D4 09EE 1788 35$: CLRL R0
E5 11 09F0 1789 BRB 20$
09F2 1791 ; Read controller parameters
09F2 1792
09F2 1793
50 01 3C 09F2 1794 40$: MOVZWL S^#SS$ NORMAL,R0 ; Assume success
51 08 3C 09F5 1795 MOVZWL S^#8,RT ; Size of P1 buffer if present
00EA 30 09F8 1796 BSBW CHECK_BUFS ; Check P1 and P2 buffers
3C A3 51 D0 09FB 1797 MOVL R1,IRPSL_MEDIA+4(R3) ; Save user P2 buffer length
3C 13 09FF 1798 BEQL 60$ ; Br if no P2 buffer
0A01 1799
3E A3 01 B0 0A01 1800 MOVW S^#SS$ NORMAL,IRPSL_MEDIA+6(R3) ; Assume success
32 A3 24 B0 0A05 1801 MOVW #LINE_PRM_BUFSIZ,IRPSW_BCNT(R3) ; Set size of required buffer
24 51 B1 0A09 1802 CMPW R1,#LINE_PRM_BUFSIZ ; If GEQU then user buffer is
0A 1E 0A0C 1803 BGEQU 50$ ; large enough
3E A3 0601 8F B0 0A0E 1804 MOVW #SS$ BUFFEROVF,IRPSL_MEDIA+6(R3) ; Return partial success
32 A3 51 B0 0A14 1805 MOVW R1,IRPSW_BCNT(R3) ; Set size to use in move
51 F684 CF 9E 0A18 1806 50$: MOVAB LINE_PARAM,R1 ; Get address of return table
54 54 DD 0A1D 1807 PUSHF R4
00A8 C5 D0 0A1F 1808 MOVL UCBSL_NO_BUFFER(R5),R4 ; Get line param block address
05 12 0A24 1809 BNEQ 52$ ; If neq buffer available
32 A3 B4 0A26 1810 CLRW IRPSW_BCNT(R3)
03 11 0A29 1811 BRB 54$
1111 30 0A2B 1812 52$: BSBW RETURN_P2 ; Return the P2 parameters
54 8ED0 0A2E 1813 54$: POPL R4
50 32 A3 B0 0A31 1814 55$: MOVW IRPSW_BCNT(R3),R0 ; Return size of buffer
50 50 10 78 0A35 1815 ASHL #16,R0,R0 ; Shift size of buffer return
50 3E A3 B0 0A39 1816 MOVW IRPSL_MEDIA+6(R3),R0 ; Set size of return
0A3D 1817
52 38 A3 D0 0A3D 1818 60$: MOVL IRPSL_MEDIA(R3),R2 ; Retrieve P1 buffer address
04 13 0A41 1819 BEQL 70$ ; Br if none
62 40 A5 7D 0A43 1820 MOVQ UCBSB_DEVCLASS(R5),(R2) ; Else, return characteristics
51 44 A5 D0 0A47 1821 70$: MOVL UCBSL_DEVDEPEND(R5),R1 ; Get device dependend info
00000000'GF 17 0A4B 1822 JMP G^EXE$FINISHIO ; Complete the I/O request
0A51 1823
```



```
0A51 1825 .SBTTL SENSE_MODEM - Perform SENSEMODE READ_MODEM FDT processing
0A51 1826 ;++
0A51 1827 ;SENSE_MODEM - Perform SENSEMODE READ_MODEM FDT processing
0A51 1828 ;
0A51 1829 ; This routine performs all FDT checking for a SENSEMODE read modem request.
0A51 1830 ; The P1 buffer is checked for write access and the request is queued to
0A51 1831 ; the driver.
0A51 1832 ;
0A51 1833 ; INPUTS:
0A51 1834 ; R3 = IRP address
0A51 1835 ; R4 = PCB address
0A51 1836 ; R5 = UCB address
0A51 1837 ; R7 = Function code
0A51 1838 ;
0A51 1839 ; OUTPUTS:
0A51 1840 ; R0 = Status for return
0A51 1841 ; R3-R5 are preserved
0A51 1842 ;--
0A51 1843 SENSE_MODEM:
51 04 3C 0A51 1844 MOVZWL S^#4,R1 ; Size of P1 buffer
00BA 30 0A54 1845 BSBW CHECK_P1 ; Check access to P1
00CA 30 0A57 1846 ; (no return on no access)
03 50 E8 0A57 1847 BSBW ALLOC_P2BUF ; Allocate a P2 buffer
FA7D 31 0A5A 1848 BLBS R0,10$ ; Branch if success
0A5D 1849 BRW ABORTIO ; Else abort the request
0A60 1850
0A60 1851 ;& WHAT DO WE DO HERE????
0A60 1852
04 51 2C A3 D0 0A60 1853 10$: MOVL IRP$L_SVAPTE(R3),R1 ; Get system buffer address
A1 38 A3 D0 0A64 1854 MOVL IRP$L_MEDIA(R3),P2B_L_BUFFER(R1) ; Set user buffer VA
21 A3 04 90 0A69 1855 CLRL IRP$L_MEDIA(R3) ; Clear buffer
0A6C 1856 MOVB #NO_FC_V_READ_MODEM,IRP$B_NOFUNC(R3) ; Set function
0A70 1857 ;BRB QUEPKT ; Give request o driver
0A70 1858
0A70 1859
0A70 1860
0A70 1861 ; I/O request packet to driver
0A70 1862
0A70 1863 QUEPKT:
00000000'GF 16 0A70 1864 SETIPL UCBSB FIPL(R5) ; Queue packet to driver
00000000'GF 17 0A74 1865 JSB G^IOC$INITIATE ; Initiate I/O request
0A7A 1866 JMP G^EXE$QIORETURN ; Lower IPL, and return
0A80 1867
```



```
0A80 1869 .SBTTL GET_CHAR_BUFS, Get P1 and P2 characteristics buffers
0A80 1870
0A80 1871 :++
0A80 1872 : GET_CHAR_BUFS - Get P1 and P2 characteristics buffers
0A80 1873
0A80 1874 : Functional description:
0A80 1875
0A80 1876 : This routine saves the P1 and P2 buffers for later use by the driver.
0A80 1877 : The P1 buffer is saved in the IRP (IRP$!_MEDIA) as a quadword value.
0A80 1878 : The P2 buffer is saved by allocating the appropriate amount of memory from
0A80 1879 : non-paged pool. The user's quota is checked before the allocation is made.
0A80 1880 : And the non-paged pool buffer is charged against the user's quota. The P2
0A80 1881 : system buffer address is passed in IRP$!_SVAPTE of the IRP.
0A80 1882
0A80 1883
0A80 1884 : INPUTS:
0A80 1885 : R3 = IRP address
0A80 1886 : R4 = PCB address
0A80 1887 : R5 = UCB address
0A80 1888
0A80 1889 : OUTPUTS:
0A80 1890 : R0 = status of buffers
0A80 1891 : R3-R5 are preserved.
0A80 1892
0A80 1893 :--
0A80 1894 GET_CHAR_BUFS: ; Get characteristics buffers
0A80 1895
0A80 1896 : Check access to P1 buffer and save P1 characteristics
0A80 1897
38 A3 7C 0A80 1898 CLRQ IRP$!_MEDIA(R3) ; Reset P1 chars
52 6C D0 0A83 1899 MOVL P1(AP),R2 ; Get address of P1 char buf
11 13 0A86 1900 BEQL 10$ ; Branch if no P1 buffer
50 0C 3C 0A88 1901 MOVZWL #SS$!_ACCVIO,R0 ; Assume access violation
0A8B 1902 IFNORD #8,(R2),20$ ; Check access
38 A3 62 7D 0A91 1903 MOVQ (R2),IRP$!_MEDIA(R3) ; Save P1 characteristics
38 A3 01 90 0A95 1904 MOVB #1,IRP$!_MEDIA(R3) ; Indicate valid P1 buffer
0A99 1905
0A99 1906 : Check access to P2 buffer and check process's buffer quota
0A99 1907
52 04 AC D0 0A99 1908 10$: MOVL P2(AP),R2 ; Get address P2 char buf desc
42 13 0A9D 1909 BEQL 40$ ; Br if no P2 buffer
50 0C 3C 0A9F 1910 MOVZWL #SS$!_ACCVIO,R0 ; Assume access violation
0AA2 1911 IFNORD #8,(R2),20$ ; Check access to descriptor
51 62 3C 0AA8 1912 MOVZWL (R2),R1 ; Get buffer length in bytes
51 01 CA 0AAB 1913 BICL #1,R1 ; Must be multiple of 2 bytes
31 13 0AAE 1914 BEQL 40$ ; Br if size is zero
52 04 A2 D0 0AB0 1915 MOVL DSC$!_POINTER(R2),R2 ; Get buffer address
50 52 D0 0AB4 1916 MOVL R2,R0 ; Copy buffer address
0C BB 0AB7 1917 PUSHR #^M<R2,R3> ; Save R2, R3
00000000'GF 16 0AB9 1918 JSB G^EXE$WRITECHKR ; Check entire buffer
06 50 E9 0ABF 1919 BLBC R0,15$ ; Branch if error
00000000'GF 16 0AC2 1920 JSB G^EXE$BUFQUOPRC ; Check for buffered quota
0C BA 0AC8 1921 15$: POPR #^M<R2,R3> ; Restore R2, R3
01 50 E8 0ACA 1922 BLBS R0,30$ ; Branch if quota ok
05 0ACD 1923 20$: RSB ; Return
0ACE 1924
0ACE 1925
```



```

; Quota OKAY, allocate buffer and copy info.
OACE 1926
OACE 1927
OACE 1928 30$: BSBW ALLOC P2BUF ; Allocate buffer
E9 OAD1 1929 BLBC R0,50$ ; Br if error
DO OAD4 1930 MOVL IRP$!_SVAPTE(R3),R0 ; Get P2 buffer address
BB OAD8 1931 PUSHF #^M<R3,R4,R5> ; Save sacred registers
28 OADA 1932 MOVC3 R1,(R2),P2B T_DATA(R0) ; Save P2 char buffer
BA OADF 1933 POPR #^M<R3,R4,R5> ; Restore registers
3C OAE1 1934 40$: MOVZWL S^#SS$_NORMAL,R0 ; Set success
05 OAE4 1935 50$: RSB ; Return

```



```

OAE5 1937      .SBTTL CHECK_BUFS, Check P1 and P2 buffers for write access
OAE5 1938
OAE5 1939      :++
OAE5 1940      : CHECK_BUFS - Check P1 and P2 buffers for write access
OAE5 1941      :
OAE5 1942      : Functional description:
OAE5 1943      :
OAE5 1944      : This routines checks the P1 and P2 buffers for write access if supplied.
OAE5 1945      :
OAE5 1946      : INPUTS:
OAE5 1947      : R1 = Size of P1 buffer needed for write access
OAE5 1948      : R3 = IRP address
OAE5 1949      : R4 = PCB address
OAE5 1950      : R5 = UCB address
OAE5 1951      : R7 = Function code
OAE5 1952      : R9 = CDB address
OAE5 1953      :
OAE5 1954      : OUTPUTS:
OAE5 1955      : R1 = Length of P2 buffer (zero if no P2 buffer)
OAE5 1956      : R2 = Address of P2 buffer in user's process space
OAE5 1957      : R0-R1 are destroyed.
OAE5 1958      :
OAE5 1959      : No RETURN on NO ACCESS
OAE5 1960      :
OAE5 1961      : IMPLICIT OUTPUTS:
OAE5 1962      :
OAE5 1963      : IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
OAE5 1964      :
OAE5 1965      :--
OAE5 1966      : CHECK_BUFS:
OAE5 1967      : BSBB CHECK_P1 ; Check P1 buffer
OAE7 1968      : CHECK_P2:
OAE7 1969      : CLRL R1 ; Assume no P2 buffer desc
52 04 AC D4 OAE7 1970      : MOVL P2(AP),R2 ; Get address of P2 desc
1B 13 OAE9 1971      : BEQL 10$ ; Br if no P2
OAEF 1972      : IFNORD #8,(R2),ACCESS ; Br if no access
51 62 3C OAF5 1973      : MOVZWL (R2),R1 ; Get length of buffer
51 01 CA OAF8 1974      : BICL #1,R1 ; Must be multiple of 2 bytes
OD 13 OAFB 1975      : BEQL 10$ ; Br if zero
50 04 A2 D0 OAFD 1976      : MOVL DSC$A_POINTER(R2),R0 ; Get buffer address
00000000'GF 16 OB01 1977      : JSB G^EXE$READCHK ; Check write access to buffer
OB07 1978      : ; (no return no access)
OB07 1979      : ; Also sets IRP$V_FUNC in IRP
52 50 D0 OB07 1980      : MOVL R0,R2 ; Copy buffer address
05 05 OB0A 1981      : RSB ; Return to caller
OB0B 1982      :
50 0C 3C OB0B 1983      : ACCESS: MOVZWL #SS$ ACCVIO,R0 ; Return access violation
F9CC 31 OB0E 1984      : BRW ABORTIO ; Abort the I/O request

```



```

OB11 1986 .SBTTL CHECK_P1, Check P1 buffer address for write access
OB11 1987
OB11 1988 ;++
OB11 1989 ; CHECK_P1 - Check P1 buffer address for write access
OB11 1990 ;
OB11 1991 ; Functional description:
OB11 1992 ;
OB11 1993 ; This routine checks the P1 buffer and if okay, the buffer address
OB11 1994 ; is saved in IRP$L_MEDIA of the IRP.
OB11 1995 ;
OB11 1996 ; INPUTS:
OB11 1997 ; R1 = Size of buffer for write access
OB11 1998 ; R3 = IRP address
OB11 1999 ; R4 = PCB address
OB11 2000 ; R5 = UCB address
OB11 2001 ; R7 = Function code
OB11 2002 ; R9 = CDB address
OB11 2003 ;
OB11 2004 ; OUTPUTS:
OB11 2005 ; R0 is destroyed.
OB11 2006 ;
OB11 2007 ; No RETURN on NO ACCESS.
OB11 2008 ;
OB11 2009 ; IMPLICIT OUTPUTS:
OB11 2010 ;
OB11 2011 ; IRP$L_MEDIA(R3) = User P1 buffer address.
OB11 2012 ; IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
OB11 2013 ;
OB11 2014 ; --
OB11 2015 CHECK_P1:
38 A3 D4 OB11 2016 CLRL IRP$L_MEDIA(R3) ; Assume no P1 buffer
50 6C D0 OB11 2017 MOVL P1(AP),R0 ; Get address of user buffer
0A 13 OB11 2018 BEQL 10$ ; Br if none
00000000'GF 16 OB11 2019 JSB G^EXE$READCHK ; Check access to buffer
38 A3 50 D0 OB11 2020 ; (No return - no access)
05 OB11 2021 MOVL R0,IRP$L_MEDIA(R3) ; Save P1 buffer address in IRP
OB23 2022 10$: RSB ; Return to caller

```



```
OB24 2024 .SBTTL ALLOC_P2BUF, Allocate a P2 buffer and charge user's quota
OB24 2025
OB24 2026 :++
OB24 2027 : ALLOC_P2BUF - Allocate a P2 buffer and charge user's quota
OB24 2028
OB24 2029 : Functional description:
OB24 2030
OB24 2031 : This routine allocates a system buffer and returns the address in the IRP at
OB24 2032 : IRP$L_SVAPTE. The size of the allocation, including buffer header must
OB24 2033 : be at least 24 bytes in length.
OB24 2034
OB24 2035 : INPUTS:
OB24 2036 : R1 = Size of allocation desired
OB24 2037 : R3 = IRP address
OB24 2038
OB24 2039 : OUTPUTS:
OB24 2040 : R0 = status of request
OB24 2041 : R1-R5 are preserved.
OB24 2042
OB24 2043 : IMPLICIT OUTPUTS:
OB24 2044 : IRP$L_SVAPTE(R3) = address of system buffer
OB24 2045 : IRP$W_BOFF(R3) = byte count charged to user's process
OB24 2046 : IRP$W_BCNT(R3) = original byte count requested
OB24 2047
OB24 2048 : All parts of the P2 buffer header are initialized, except for the
OB24 2049 : user's P2 buffer address.
OB24 2050
OB24 2051 :--
OB24 2052 ALLOC_P2BUF:
OB24 2053 TSTL R1 ; Allocate a non-paged buffer
OB24 2054 BEQL 30$ ; Zero length buffer?
OB24 2055 PUSHF #^M<R1,R2,R3> ; Br if yes
OB24 2056 MOVW R1,IRP$W_BCNT(R3) ; Save registers
OB24 2057 CMPL R1,S^#24-P2B_C_LENGTH ; Save original byte count
OB24 2058 BGTRU 5$ ; Is buffer big enough?
OB24 2059 MOVL S^#24-P2B_C_LENGTH,R1 ; Br if yes
OB24 2060 5$: ADDL2 S^#P2B_C_LENGTH,R1 ; Else, set size to minimum
OB24 2061 JSB G^EXES$BUFQUOPRC ; Add in size of header
OB24 2062 BLBC R0,10$ ; Check for buffered quota
OB24 2063 ; Branch if quota bad
OB24 2064 ; Quota OKAY, allocate buffer and copy info.
OB24 2065
OB24 2066 PUSHF R1 ; Save size to charge user
OB24 2067 JSB G^EXES$ALONONPAGED ; Go allocate a buffer
OB24 2068 BLBS R0,20$ ; Br if success
OB24 2069 TSTL (SP)+ ; Pop saved size
OB24 2070 10$: POPF #^M<R1,R2,R3> ; Restore registers
OB24 2071 RSB ; Return with error code in R0
OB24 2072
OB24 2073 ; System buffer allocated decrement user's quota
OB24 2074
OB24 2075 20$: POPL R3 ; Restore user quota charge
OB24 2076 MOVAB P2B_T_DATA(R2),P2B_L_POINTER(R2) ; Set address to start of data
OB24 2077 MOVW R3,P2B_W_SIZE(R2) ; Save buffer size in buffer
OB24 2078 MOVB S^#DYN$C_BUFIO,P2B_B_TYPE(R2) ; Set structure type
OB24 2079 MOVL R2,R0 ; Save P2 char buf addr
OB24 2080 MOVL PCB$L_JIB(R4),R2 ; Get JIB address
```

32 A3 51 D5 50 13 0E BB 51 B0 51 D1 03 1A 51 OC D0 51 OC C0 00000000 GF 16 0D 50 E9

00000000 GF 16 05 50 E8 8E D5 0E BA 05

62 OC 53 8E D0 08 A2 53 9E 0A A2 53 B0 50 13 90 50 52 D0 52 0080 C4 D0

NODRIVER
V04-001

H 16
- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 49
ALLOC_P2BUF, Allocate a P2 buffer and c 6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (34)

20	A2	53	C2	0B69	2081	SUBL	R3,JIB\$ _L BYTCNT(R2)	; Decrement user's quota
		0E	BA	0B6D	2082	POPR	#^M<R1,R2,R3>	; Restore registers
2C	A3	50	D0	0B6F	2083	MOVL	R0,IRP\$ _L SVAPTE(R3)	; Save P2 buffer address in IRP
30	A3	08	B0	0B73	2084	MOVW	P2B_W SIZE(R0),IRP\$W_BOFF(R3)	; Return buffer size in IRP
	50	01	3C	0B78	2085	MOVZWL	S^#SS\$ ₋ NORMAL,R0	; Set success
			05	0B7B	2086	RSB		; Return to caller

```
OB7C 2088 .SBTTL CHG_UCB_NOB, Change UCB and the NOB parameter values
OB7C 2089
OB7C 2090 :++
OB7C 2091 : CHG_UCB_NOB - Change UCB and the NOB parameter values
OB7C 2092 :
OB7C 2093 : Functional description:
OB7C 2094 :
OB7C 2095 : This routine is called to initialize the NOB with new P1 and P2 buffer
OB7C 2096 : characteristics. It is assumed here that the parameters have already
OB7C 2097 : been validated.
OB7C 2098 :
OB7C 2099 : INPUTS:
OB7C 2100 :
OB7C 2101 : R3 = IRP address
OB7C 2102 : R5 = UCB address
OB7C 2103 : R6 = Protocol mode in which to run the driver
OB7C 2104 :
OB7C 2105 : IPL = FIPL
OB7C 2106 :
OB7C 2107 : OUTPUTS:
OB7C 2108 : R0-R2 = destroyed.
OB7C 2109 :
OB7C 2110 :--
OB7C 2111 CHG_UCB_NOB:
OB7C 2112 PUSH  R4 ; Save R4
54 00A8 C5 DD OB7E 2113 MOVL  UCB$NO_BUFFER(R5),R4 ; Get NOB address
33 38 A3 E9 OB83 2114 BLBC  IRP$MEDIA(R3),10$ ; Br if no P1 buffer
OB87 2115
OB87 2116 ; Set new P1 buffer characteristics
OB87 2117
OB87 2118 BBS  #NO_DS_V_INITED,- ; Br if device inited
0A 68 A5 E0 OB89 2119 UCB$W_DEVSTS(R5),5$
OB8C 2120
OB8C 2121 ; Please note that setting this of this parameter on device startup is
OB8C 2122 ; usually accomplished via the NOB$W_DEVBUFSIZ field in the NOB buffer. This
OB8C 2123 ; was done to make the validating and the setting of this field simpler.
OB8C 2124
42 A5 3A A3 B0 OB8C 2125 MOVW  IRP$MEDIA+2(R3),UCB$W_DEVBUFSIZ(R5) ; Set new buffer size
74 A4 42 A5 B0 OB91 2126 MOVW  UCB$W_DEVBUFSIZ(R5),NOB$W_DEVBUFSIZ(R4) ; Make sure both are updated
OB96 2127
44 A5 18 08 00 F0 OB96 2128 5$: INSV  #0,#8,#24,UCB$DEVDEPEND(R5) ; Reset all Read/Write flags
3C A3 C8 OB9C 2129 BISL  IRP$MEDIA+4(R3),- ; Set new characteristics
44 A5 OB9F 2130 UCB$DEVDEPEND(R5)
OBA1 2131
OBA1 2132 ; Now update UCB based on P1 buffer
OBA1 2133
OBA1 2134 ASSUME NMASC_LINPR_POI EQ 0
OBA1 2135
70 A4 94 OBA1 2136 CLRB  NOB$B_PRO(R4) ; Assume point to point mode
OBA4 2137
OBA4 2138 ASSUME NMASC_LINCN_NOR EQ 0
OBA4 2139
72 A4 94 OBA4 2140 6$: CLRB  NOB$B_CON(R4) ; Assume normal mode
03 44 A5 E1 OBA7 2141 BBC  #XMSV_CHR_LOOPB,- ; Br if not loopback mode
OBA9 2142 UCB$DEVDEPEND(R5),7$
OBAC 2143 ASSUME NMASC_LINCN_LOO EQ 1
72 A4 96 OBAC 2144 INCB  NOB$B_CON(R4) ; Must be loopback mode
```



```

71 A4 94 OBAF 2145 7$: ASSUME NMA$C_DPX_FUL EQ 0
03 44 A5 02 E1 OBAF 2146 CLR$B NOB$B_DUP(R4) ; Assume full duplex
OBB2 2147 BBC #XMSV_CHR_HDPLX,- ; Br if not half duplex
OBB4 2148 UCB$C_DEVDEPEND(R5),10$
OBB7 2149 ASSUME NMA$C_DPX_HAL EQ 1
71 A4 96 OBB7 2150 INCB NOB$B_DUP(R4) ; Must be half duplex
OBB8 2151
OBB9 2152
OBBA 2153 ; Set new P2 buffer characteristics
OBBB 2154
OBBB 2155 10$: MOVAB LINE_PARAM,R1 ; Get address of verification table
51 F4E2 CF 9E OBBB 2156 BSBW UPDATE_P2 ; Update the UCB
OF0C 30 OBC2 2157
OBC2 2158 ; Please note that setting this of this parameter on device startup is
OBC2 2159 ; usually accomplished via the NOB$W_DEVBUFSIZ field in the NOB buffer. This
OBC2 2160 ; was done to make the validating and the setting of this field simpler.
OBC2 2161
42 A5 74 A4 B0 OBC2 2162 MOVW NOB$W_DEVBUFSIZ(R4),UCB$W_DEVBUFSIZ(R5) ; Set def buffer size
OBC7 2163
OBC7 2164 ; Set device characteristics and device mode definition
OBC7 2165
44 A5 94 OBC7 2166 CLR$B UCB$C_DEVDEPEND(R5) ; Reset all characteristics
OBCA 2167 20$: SETBIT #XMSV_CHR_HDPLX,UCB$C_DEVDEPEND(R5) ; Assume half duplex mode
OBCF 2168 ASSUME NMA$C_DPX_FUL EQ 0
71 A4 95 OBCF 2169 TSTB NOB$B_DUP(R4) ; Full duplex mode?
05 12 OBD2 2170 BNEQ 30$ ; Br if no - okay
OBD4 2171 CLRBIT #XMSV_CHR_HDPLX,UCB$C_DEVDEPEND(R5) ; Set to full duplex mode
OBD9 2172 30$: ASSUME NMA$C_LINCN_NOR EQ 0
72 A4 95 OBD9 2173 TSTB NOB$B_CON(R4) ; Is line in normal mode?
05 13 OBDC 2174 BEQL 40$ ; Br if yes
OBDE 2175 SETBIT #XMSV_CHR_LOOPB,UCB$C_DEVDEPEND(R5) ; Set loopback mode
54 8ED0 OBE3 2176 40$: POPL R4 ; Restore R4
05 OBE6 2177 RSB
OBE7 2178
```



```
OBE7 2180
OBE7 2181      .SBTTL CHG_TRIB, Change trib parameter values
OBE7 2182
OBE7 2183 :++
OBE7 2184 : CHG_TRIB - Change trib parameter values
OBE7 2185 :
OBE7 2186 : Functional description:
OBE7 2187 :
OBE7 2188 : This routine is called to initialize the trib parameter block with
OBE7 2189 : new P1 and P2 buffer characteristics. It is assumed here that the
OBE7 2190 : parameters have already been validated.
OBE7 2191 :
OBE7 2192 : INPUTS:
OBE7 2193 :       R3 = IRP address
OBE7 2194 :       R5 = UCB address
OBE7 2195 :
OBE7 2196 : IPL = FIPL
OBE7 2197 :
OBE7 2198 : OUTPUTS:
OBE7 2199 :       R1,R2 = destroyed.
OBE7 2200 :
OBE7 2201 :--
OBE7 2202 CHG_TRIB:
OBE7 2203      PUSHL R4 ; Validate P2 buffer
OBE7 2204      MOVL  UCB$NO_BUFFER(R5),R4 ; Save R4
OBE7 2205      MOVAB TRIB_PARAM,R1 ; Get NOB address
OBE7 2206      MOVAB NOB$DDCMP(R4),R4 ; Get address of verify table
OBE7 2207      BLBC  IRP$MEDIA(R3),10$ ; Get the addr of param block
OBE7 2208      ; Br if no P1 buffer
OBE7 2209      ; Set new P1 buffer characteristics
OBE7 2210
OBE7 2211      BISL  IRP$MEDIA+4(R3),UCB$DEVDEPEND(R5) ; Set new char
OBE7 2212
OBE7 2213      ASSUME NMA$STATE_ON EQ 0
OBE7 2214      ASSUME NMA$STATE_OFF EQ 1
OBE7 2215
OBE7 2216      CLRB  DLK$B_MAINT(R4) ; Assume MOP
OBE7 2217      BBS   #XMSV_CHR_MOP,UCB$DEVDEPEND(R5),10$ ; Branch BS if true
OBE7 2218      INCB  DLK$B_MAINT(R4) ; Set NORMAL mode
OBE7 2219
OBE7 2220      ; Set new P2 buffer characteristics
OBE7 2221
OBE7 2222      10$: BSBW  UPDATE_P2 ; Update trib parameters
OBE7 2223      CLRBIT #XMSV_CHR_MOP,UCB$DEVDEPEND(R5) ; Assume NORMAL mode
OBE7 2224      BLBS  DLK$B_MAINT(R4),20$ ; Branch LBS if true
OBE7 2225      SETBIT #XMSV_CHR_MOP,UCB$DEVDEPEND(R5) ; Set MOP mode
OBE7 2226      20$: POPL  R4 ; Restore R4
OBE7 2227      RSB ; Return to caller
OBE7 2228
```

54 00A8 C5 DD OBE7 2203
51 F4E2 CF DO OBE9 2204
54 60 A4 9E OBE7 2205
10 38 A3 E9 OBF3 2206
OBF7 2207
OBF8 2208
OBF8 2209
OBF8 2210
44 A5 3C A3 C8 OBF8 2211
OC00 2212
OC00 2213
OC00 2214
OC00 2215
03 44 A5 08 A4 94 OC00 2216
08 00 E0 OC03 2217
08 A4 96 OC08 2218
OC0B 2219
OC0B 2220
OC0B 2221
OEC0 30 OC0B 2222
05 08 A4 E8 OC0E 2223
OC13 2224
OC17 2225
54 8ED0 OC1C 2226
05 OC1F 2227
OC20 2228


```

OC20 2230 .SBTTL NOS$STARTIO - Start setmode I/O operation
OC20 2231 :++
OC20 2232 : NOS$STARTIO - Start setmode operation
OC20 2233 :
OC20 2234 : FUNCTIONAL DESCRIPTION:
OC20 2235 :
OC20 2236 : This routine is entered to process a setmode request. All setmode requests
OC20 2237 : are queued through the UCB to single-stream them.
OC20 2238 :
OC20 2239 : SETMODE FUNCTIONS --
OC20 2240 :
OC20 2241 : For all functions a change in the characteristics is done.
OC20 2242 :
OC20 2243 : For control startup, the UCB is initialized, and the DDCMP timer is
OC20 2244 : started.
OC20 2245 : For trib startup, the protocol is set up with characteristics and
OC20 2246 : started, and transmits and receives are started on the board.
OC20 2247 :
OC20 2248 : For control shutdown, the DDCMP timer is stopped,
OC20 2249 : all quotas are returned; and a call is made to trib shutdown to shut
OC20 2250 : the trib.
OC20 2251 : For trib shutdown, all buffers and IRP's are returned,
OC20 2252 : and the protocol is halted.
OC20 2253 :
OC20 2254 : INPUTS:
OC20 2255 : R3 = I/O packet address
OC20 2256 : R5 = UCB address
OC20 2257 :
OC20 2258 : OUTPUTS:
OC20 2259 : R3 and R5 preserved.
OC20 2260 :
OC20 2261 :--
OC20 2262 :.ENABL LSB
OC20 2263 NOS$STARTIO:
51 21 A3 9A OC20 2264 MOVZBL IRP$B_NOFUNC(R3),R1 ; Start I/O routine
OC20 2265 PUSHBL R4 ; Get the function code
OC20 2266 BSBB START_DISP ; Branch to case
51 44 A5 8ED0 OC28 2267 POPL R4
OC2B 2268 MOVL UCB$L_DEVDEPEND(R5),R1 ; Set device characteristics
OC2F 2269 REQCOM ; Complete the request
OC35 2270
OC35 2271 START_DISP:
OC35 2272 $DISPATCH R1,TYPE=B,-
OC35 2273 <- ; Function
OC35 2274 <NO_FC_V_STRT_CIR Action
OC35 2275 <NO_FC_V_STRT_LIN START_CIRCUIT>,-
OC35 2276 <NO_FC_V_STOP_CIR SHUTDOWN_CIRCUIT>,-
OC35 2277 <NO_FC_V_STOP_LIN SHUTDOWN_LINE>,-
OC35 2278 <NO_FC_V_READ_MODEM READ_MODEM>,-
OC35 2279 >
OC43 2280
OC43 2281 5$: BUG_CHECK NOBUFPCKT,FATAL ; Anything else is fatal
OC47 2282
OC47 2283 .DSABL LSB
OC47 2284

```



```
OC47 2286 .SBTTL START_TRANSMIT - Start transmit I/O
OC47 2287 :++
OC47 2288 : START_TRANSMIT - Start transmit I/O
OC47 2289 :
OC47 2290 : FUNCTIONAL DESCRIPTION:
OC47 2291 :
OC47 2292 : This routine is entered to start transmitting over the async port driver. In
OC47 2293 : order to do this the transmit is split into several pieces. The header, the
OC47 2294 : header CRC, data, data CRC, and the PADS. It works by checking to see if
OC47 2295 : the transmitter is busy. The transmitter is busy if the UCB$V_INT bit is set in
OC47 2296 : UCB$W_STS. If the port is not busy then a transmit is started by calling
OC47 2297 : the PORT_STARTIO routine and the UCB$V_INT is set in UCB$W_STS. The transmit
OC47 2298 : finishes in GETNXT routine after all the parts of the message has been
OC47 2299 : transmitted. If the device is running half duplex then transmits will only be
OC47 2300 : sent until the select flag is sent.
OC47 2301 :
OC47 2302 : INPUTS:
OC47 2303 : R1 = Status information from routine calling START_TRANSMIT
OC47 2304 : More specifically it is an interface between the protocol
OC47 2305 : timer routine and this routine to tell start transmit that
OC47 2306 : the protocol timer has expired therefore send the message.
OC47 2307 : R4 = NOB address
OC47 2308 : R5 = UCB address
OC47 2309 :
OC47 2310 : OUTPUTS:
OC47 2311 : NONE
OC47 2312 :
OC47 2313 : --
OC47 2314 : START_TRANSMIT:
OC47 2315 :
OC47 2316 : ; If the DDCMP timer has expired a message may need to be sent
OC47 2317 :
OC47 2318 BBS #DLK$V_TMREXPD,R1,20$
OC47 2319 10$: BBS #NO_DS_V_XMTING,- ; If BS then the xmt'r is off
OC47 2320 UCB$W_DEVSTS(R5),25$
OC47 2321 20$: DSBINT UCB$B_DIPL(R5)
OC47 2322 BBSS #UCB$V_INT,UCB$W_STS(R5),30$ ; If BS then port is busy
OC47 2323 TSTB NOB$B_XSTATE(R4) ; If NEQ then the transmitter is bus
OC47 2324 BNEQ 30$
OC47 2325 BSBW GET_XMT ; Get next message to send
OC47 2326 ENBINT
OC47 2327 BLBC R0,40$ ; If LBC no message to send
OC47 2328 MOVL UCB$B_TT_PORT(R5),R2 ; Get port vector table
OC47 2329 MOVAL XMTQ$B_MSGHDR(R8),UCB$B_TT_OUTADR(R5) ; Set address of block to send
OC47 2330 MOVZBW #NOSC_HEADER_LEN,UCB$W_TT_OUTLEN(R5) ; Set size of block to send
OC47 2331 PUSH R4
OC47 2332 DSBINT UCB$B_DIPL(R5)
OC47 2333 MOVB #NOSC_HEADER,NOB$B_XSTATE(R4) ; Set transmitter state
OC47 2334
OC47 2335 ; The following instruction must be the last instruction executed before
OC47 2336 ; the return to the port output routine. The reason for this is that the
OC47 2337 ; condition codes must correctly so that the port does the right transfer
OC47 2338
OC47 2339 MNEGB #1,UCB$B_TT_OUTYPE(R5) ; Set block transfer to port
OC47 2340 JSB @PORT_STARTIO(R2) ; Call at port startio routine
OC47 2341 ENBINT
OC47 2342 POPL R4
```

05 51 0F E0 OC47 2318 BBS #DLK\$V_TMREXPD,R1,20\$
45 68 A5 E0 OC47 2319 10\$: BBS #NO_DS_V_XMTING,- ; If BS then the xmt'r is off
3D 64 A5 01 E2 OC47 2320 UCB\$W_DEVSTS(R5),25\$
42 A4 95 OC47 2321 20\$: DSBINT UCB\$B_DIPL(R5)
38 12 OC47 2322 BBSS #UCB\$V_INT,UCB\$W_STS(R5),30\$; If BS then port is busy
007B 30 OC47 2323 TSTB NOB\$B_XSTATE(R4) ; If NEQ then the transmitter is bus
38 50 E9 OC47 2324 BNEQ 30\$
52 0118 C5 D0 OC47 2325 BSBW GET_XMT ; Get next message to send
011C C5 24 A8 DE OC47 2326 ENBINT
0120 C5 06 9B OC47 2327 BLBC R0,40\$; If LBC no message to send
54 DD OC47 2328 MOVL UCB\$B_TT_PORT(R5),R2 ; Get port vector table
42 A4 01 90 OC47 2329 MOVAL XMTQ\$B_MSGHDR(R8),UCB\$B_TT_OUTADR(R5) ; Set address of block to send
OC47 2330 MOVZBW #NOSC_HEADER_LEN,UCB\$W_TT_OUTLEN(R5) ; Set size of block to send
OC47 2331 PUSH R4
OC47 2332 DSBINT UCB\$B_DIPL(R5)
OC47 2333 MOVB #NOSC_HEADER,NOB\$B_XSTATE(R4) ; Set transmitter state
OC47 2334
OC47 2335 ; The following instruction must be the last instruction executed before
OC47 2336 ; the return to the port output routine. The reason for this is that the
OC47 2337 ; condition codes must correctly so that the port does the right transfer
OC47 2338
010B C5 01 8E OC47 2339 MNEGB #1,UCB\$B_TT_OUTYPE(R5) ; Set block transfer to port
00 B2 16 OC47 2340 JSB @PORT_STARTIO(R2) ; Call at port startio routine
54 8ED0 OC47 2341 ENBINT
OC47 2342 POPL R4


```
50 01 3C 0C95 2343 25$: MOVZWL S^#SS$_NORMAL,R0
05 0C98 2344 RSB
0C99 2345
0C99 2346 30$: CLRBIT #UCB$V_INT,UCB$W_STS(R5) ; Reset port in use
0C9E 2347 ENBINT
50 01 3C 0CA1 2348 MOVZWL S^#SS$_NORMAL,R0
05 0CA4 2349 RSB
0CA5 2350
0CA5 2351 40$: DSBINT UCB$B_DIPL(R5)
0CAC 2352 CLRBIT #UCB$V_INT,UCB$W_STS(R5) ; Clear nothing given to port
0CB1 2353 ENBINT
04 68 A5 09 E5 0CB4 2354 BBCC #NO_DS_V_MSG_SENT,UCB$W_DEVSTS(R5),50$ ; If BC do not try to send an
50 01 3C 0CB9 2355 MOVZWL S^#SS$_NORMAL,R0
05 0CBC 2356 RSB
0CBD 2357
0CBD 2358
0CBD 2359 50$: PUSHR #^M<R4,R5>
56 30 BB 0CBF 2360 MOVZBL #DLK$C_XMTMSG,R6 ; Inform DDCMP that the last
57 02 9A 0CC2 2361 MOVZBL #DLK$M_QEMPTY,R7 ; msg on queue was sent
55 0C A4 D0 OCC5 2362 MOVL NOB$A_PRO_BUFFER(R4),R5 ; Get addr of start of TFB
F334 30 OCC9 2363 BSBW DDCMP ; Branch to protocol
30 BA OCCC 2364 POPR #^M<R4,R5> ; Restore registers
OCCE 2365
OCCE 2366 ASSUME DLK$V_MSGSENT EQ 0
56 02 91 OCCE 2367 CMPB #DLK$C_XMTMSG,R6 ; If NEQ no ACK to be xmted
08 12 OCD1 2368 BNEQ 60$
05 57 E9 OCD3 2369 BLBC R7,60$ ; If BC no ACK need be sent
51 D4 OCD6 2370 CLRL R1 ; Set no status info
FF6C 31 OCD8 2371 BRW START_TRANSMIT ; Else send the ACK
50 01 3C OCDB 2372 60$: MOVZWL S^#SS$_NORMAL,R0 ; Set status
05 OCDE 2373 RSB
OCDF 2374
```



```
OCDF 2376 .SBTTL GET_XMT - Get next transmit ti send
OCDF 2377 :++
OCDF 2378 : GET_XMT - Get next transmit to send
OCDF 2379 :
OCDF 2380 : FUNCTIONAL DESCRIPTION:
OCDF 2381 :
OCDF 2382 : This routine checks the various tranmit queues and sets up the device with
OCDF 2383 : the next message to transmit. This means that it increments the transmit
OCDF 2384 : count and decides if the transmitter must be shut down after this message
OCDF 2385 : is sent (for half duplex only). You will notice a hack pertaining to the
OCDF 2386 : control message buffer. The reason for this is that DDCMP makes the
OCDF 2387 : assumption that it can change the data inside the control messages at
OCDF 2388 : will. There is no problem with that for most devices, however for AYSNCH
OCDF 2389 : DDCMP we calculate our own CRC's and thus if data gets changed from under
OCDF 2390 : us we will lose the message (in a most inefficient way we should note).
OCDF 2391 :
OCDF 2392 : INPUTS:
OCDF 2393 : R4 = NOB address
OCDF 2394 : R5 = UCB address
OCDF 2395 :
OCDF 2396 : IPL = DIPL
OCDF 2397 :
OCDF 2398 : OUTPUTS:
OCDF 2399 : R0 = status LBS then message to send, LBC no message to send
OCDF 2400 : R4,R5 are preserved
OCDF 2401 : R8 = address of message to send
OCDF 2402 :
OCDF 2403 :--
OCDF 2404 : GET_XMT:
OCDF 2405 : CLRL R0 ; Assume no message to send
56 OC A4 D4 OCE1 2406 : MOVL NOB$A_PRO_BUFFER(R4),R6 ; Get protocol buffer address
51 20 B6 OF OCE5 2407 : REMQUE @TF$Q_CTLQ(R6),R1 ; Check control queue
OCDF 2408 : BVC 50$ ; If VC then messae to send
58 30 B6 OF OCEB 2409 : REMQUE @TF$Q_XMTQ(R6),R8 ; Check data queue
OCDF 2410 : BVS 40$ ; If VS then no message to send
OA A8 13 91 OCF1 2411 : CMPB S^#DYN$C_BUFIO,XMTQ$B_BUFTYP(R8) ; If NEQ then a problem with
OCDF 2412 : BNEQ 60$ ; queues
03 1F A8 02 E0 OCF7 2413 10$: BBS #XMTQ$V_CONTROL,XMTQ$B_FLAG(R8),20$ ; If BS then cntrl msg don't inc
OCDF 2414 : INCB TFSB_XCNT(R6)
OA 1F A8 10 A6 96 OCF8 2415 20$: BBCC #XMTQ$V_SELECT,XMTQ$B_FLAG(R8),30$ ; If BC do not turn link
OCDF 2416 : BBC #XMS$V_CRR_HDPLX,UCB$S_DEVDEPEND(R5),30$ ; If Not half duplex dont' t
05 44 A5 02 E1 OD04 2417 : SETBIT #NO_DS_V_XMTING,UCB$W_DEVSTS(R5) ; Set xmter off, but finish xmting
OCDF 2418 30$: TSTL NOB$S_XMT_INPR(R4)
OCDF 2419 : BNEQ 60$
OCDF 2420 : MOVL R8,NOB$S_XMT_INPR(R4) ; Set up xmt inprogress
OCDF 2421 :
OCDF 2422 : Note although this may look wrong the instruction is correct because
OCDF 2423 : teh terminal line speed is really a two byte field. The low byte is the
OCDF 2424 : transmit speed and the high byte is the receive speed. If the receive
OCDF 2425 : speed is zero then it is the same as the transmit speed.
50 00F4 C5 9A OD17 2426 : MOVZBL UCB$W_TT_SPEED(R5),R0
OCDF 2427 :
OCDF 2428 : Set the due time on the transmit to be given to the port.
OCDF 2429 :
OCDF 2430 : MOVZWL DUETIM_TABLE[R0],R0
51 00000000 GF D0 OD22 2431 : MOVL G^EXE$GL_ABSTIM,R1
3C A4 51 50 C1 OD29 2432 : ADDL3 R0,R1,NOB$S_XMT_DUETIM(R4)
```


50	01	3C	0D2E	2433	SETBIT	#NO_DS_V_XMT_TIME,UCBSW_DEVSTS(R5)
		05	0D33	2434	MOVZWL	S^#SS\$-NORMAC,R0
			0D36	2435	RSB	
			0D37	2436		
58	0080	C4	DE	0D37	2437	50\$: MOVAL NOB\$Z_CTL_MSG(R4),R8 ; Get the address of buffer to use
1F	A8	1F	A1	88	0D3C	2438
	1F	A1	03	8A	0D41	2439
20	A8	20	A1	B0	0D45	2440
24	A8	24	A1	D0	0D4A	2441
28	A8	28	A1	B0	0D4F	2442
			A1	11	0D54	2443
					0D56	2444
					0D56	2445
					0D5A	2446
					60\$: BUG_CHECK	NOBUFCKT,FATAL

BISB XMTQ\$B_FLAG(R1),XMTQ\$B_FLAG(R8) ; Copy control message flags
 BICB #XMTQ\$M_SELECT,XMTQ\$M_ONQUEUE,XMTQ\$B_FLAG(R1) ; Clear flags
 MOVW XMTQ\$W_HCRC(R1),XMTQ\$W_HCRC(R8) ; Get CRC
 MOVL XMTQ\$B_MSGHDR(R1),XMTQ\$B_MSGHDR(R8) ; Get first longword of hdr
 MOVW XMTQ\$B_MSGHDR+4(R1),XMTQ\$B_MSGHDR+4(R8) ; Get second word
 BRB 10\$; Cont processing


```

OD5A 2448 .SBTTL READ_MODEM - Read device modem register
OD5A 2449 :++
OD5A 2450 :READ_MODEM - Read device modem register
OD5A 2451 :
OD5A 2452 : This routine returns the device modem register in the buffer passed.
OD5A 2453 : These are set according to the XMDEF's for modem bits.
OD5A 2454 :
OD5A 2455 : INPUTS:
OD5A 2456 : R3 = IRP address
OD5A 2457 : R5 = UCB address
OD5A 2458 :
OD5A 2459 :--
OD5A 2460 READ_MODEM:
OD5A 2461
OD5A 2462 ;& WHAT DO WE DO HERE?????
OD5A 2463
06 68 A5 06 E0 OD5A 2464 BBS #NO_DS_V_INITED,UCB$W_DEVSTS(R5),5$ ; If BS then read register
50 20D4 8F 3C OD5F 2465 MOVZWL #SS$_DEVINACT,R0 ; Else return device inactive
05 OD64 2466 RSB
OD65 2467
50 01 3C OD65 2468 5$: MOVZWL S^#SS$_NORMAL,R0
05 OD68 2469 RSB
  
```



```
0D69 2471 .SBTTL FILL DUETIME_TABLE - Set up the due time table
0D69 2472 :++
0D69 2473 : FILL_DUETIME_TABLE
0D69 2474 :
0D69 2475 : Functional description:
0D69 2476 :
0D69 2477 : This routine sets up the default values in the table as well as sets up the
0D69 2478 : known terminal line speeds based on the default buffer size due times.
0D69 2479 : The due time is the max length of time we believe it should take to
0D69 2480 : transmit or receive a given message based on the speed of the line
0D69 2481 :
0D69 2482 : INPUTS:
0D69 2483 : R5 = UCB address
0D69 2484 :
0D69 2485 : OUTPUTS:
0D69 2486 : The DUETIM_TABLE is filled.
0D69 2487 :
0D69 2488 :--
0D69 2489 FILL_DUETIM_TABLE:
07 BB 0D69 2490 PUSHRR #^M<R0,R1,R2>
0D6B 2491
0D6B 2492 ; Set up table with defaults. This is in case a terminal line supports a
0D6B 2493 ; speed that we don't know about. We wish to make this reasonable so we will
0D6B 2494 ; wait at least one second.
0D6B 2495
50 0100 8F 3C 0D6B 2496 MOVZWL #NOSC_DUETIM_TABLE_SIZE,R0
51 D4 0D70 2497 CLRL R1
F38C CF41 01 B0 0D72 2498 10$: MOVW #1,DUETIM_TABLE[R1]
51 96 0D78 2499 INCB R1
50 F5 0D7A 2500 SOBGTR R0,10$
50 42 A5 0C A1 0D7D 2501 ADDW3 #12,UCB$W_DEVBUFSIZ(R5),R0
50 08 A4 0D82 2502 MULW2 #8,R0
0D85 2503 STORE-DUETIM 50,R0,R1,R2,DUETIM_TABLE
0D95 2504 STORE-DUETIM 75,R0,R1,R2,DUETIM_TABLE
0DA7 2505 STORE-DUETIM 110,R0,R1,R2,DUETIM_TABLE
0DB9 2506 STORE-DUETIM 134,R0,R1,R2,DUETIM_TABLE
0DCB 2507 STORE-DUETIM 150,R0,R1,R2,DUETIM_TABLE
0DD5 2508 STORE-DUETIM 300,R0,R1,R2,DUETIM_TABLE
0DEF 2509 STORE-DUETIM 600,R0,R1,R2,DUETIM_TABLE
0E01 2510 STORE-DUETIM 1200,R0,R1,R2,DUETIM_TABLE
0E13 2511 STORE-DUETIM 1800,R0,R1,R2,DUETIM_TABLE
0E25 2512 STORE-DUETIM 2000,R0,R1,R2,DUETIM_TABLE
0E37 2513 STORE-DUETIM 2400,R0,R1,R2,DUETIM_TABLE
0E49 2514 STORE-DUETIM 3600,R0,R1,R2,DUETIM_TABLE
0E5B 2515 STORE-DUETIM 4800,R0,R1,R2,DUETIM_TABLE
0E6D 2516 STORE-DUETIM 7200,R0,R1,R2,DUETIM_TABLE
0E7F 2517 STORE-DUETIM 9600,R0,R1,R2,DUETIM_TABLE
0E91 2518 STORE-DUETIM 19200,R0,R1,R2,DUETIM_TABLE
07 BA 0EA3 2519 POPR #^M<R0,R1,R2>
05 05 0EA5 2520 RSB
```

; Get the max msg size
; mult by 8 bits per byte


```
OEA6 2522 .SBTTL START - Start unit, device and/or protocol
OEA6 2523 :++
OEA6 2524 : START_LINE - Start unit
OEA6 2525 :
OEA6 2526 : Functional description:
OEA6 2527 :
OEA6 2528 : This routine initializes the UCB; allocates the map registers for
OEA6 2529 : transmits and receives; and master clears the device.
OEA6 2530 :
OEA6 2531 : If a failure occurs the line shutdown sequence is entered.
OEA6 2532 :
OEA6 2533 : INPUTS:
OEA6 2534 : R3 = I/O packet
OEA6 2535 : R5 = UCB address
OEA6 2536 :
OEA6 2537 : IMPLICIT INPUTS:
OEA6 2538 :
OEA6 2539 : IRP$ _MEDIA contains a copy of the mode buffer specified
OEA6 2540 : by the user.
OEA6 2541 : IRP$ _QUOTA contains the quota taken from the user for the unit.
OEA6 2542 :
OEA6 2543 : OUTPUTS:
OEA6 2544 : The request is completed.
OEA6 2545 :
OEA6 2546 :--
OEA6 2547 START_LINE:
OEA6 2548 BBC #NO_DS_V_INITED,- ; Start protocol operation
OEA6 2549 UCB$W_DEVSTS(R5),5$ ; If BC then device not inited
OEA6 2550 MOVZWL #SS$ _DEVACTIVE,R0 ; Branch to set up idle UCB
OEA6 2551 RSB ; Set device active
OEA6 2552 ; ... and return
OEA6 2553 5$: MOVL UCB$ _NO_BUFFER(R5),R4 ; Get NOB address
OEA6 2554 BICL #NO$C _LINE_PAR,UCB$ _DEVDEPEND(R5) ; Reset all Read/Write flags
OEA6 2555 MOVW IRP$W _QUOTA(R3),NOB$ _QUOTA(R4) ; Set quota for RCV's
OEA6 2556 CLRW IRP$W _QUOTA(R3) ; Quota is return is handled by us
OEA6 2557
OEA6 2558 MOVL IRP$ _PID(R3),NOB$ _PID(R4) ; Save starter's PID
OEA6 2559 MOVW IRP$W _CHAN(R3),NOB$ _CHANL(R4) ; Save channel number
OEA6 2560 BSBW FILL DUETIM TABLE ; Fill in xmt/rcv message due times
OEA6 2561 BISW #NO_DS_M_INITED,UCB$W_DEVSTS(R5) ; Indicate UCB initialized
OEA6 2562
OEA6 2563 ; Call the terminal port driver to set line speed and parity. This call
OEA6 2564 ; may affect the UCB$ _DEVDEPEND field, thus we must save this field
OEA6 2565 ; so that information does not get lost.
OEA6 2566
OEA6 2567 DD OED9 2567 PUSHL UCB$ _DEVDEPEND(R5)
OEA6 2568 DD OEDC 2568 MOVL UCB$ _TT_PORT(R5),R2 ; Get port vector table
OEA6 2569 DD EEE1 2569 DSBINT UCB$B _DIPL(R5)
OEA6 2570 DD EEE8 2570 MOVW #NO$C _SET_LINE,UCB$B _TT_PARITY(R5) ; We must always send 8 bits of d
OEA6 2571 DD EED 2571
OEA6 2572 ; Make sure that the device does not do XON/XOFF. If it should be doing
OEA6 2573 ; this, you can notice this because transmits appear to just stop
OEA6 2574 ; on the line.
OEA6 2575
OEA6 2576 DD EED 2576 CLRBIT #TTY$V_PC_XOFENA,UCB$W _TT_PRTCTL(R5)
OEA6 2577 DD EEF3 2577
OEA6 2578 ; Set the port up so that it takes advantage of the DMA capabilities
```



```
0122 0B A8 0EF3 2579 ; of the device. Devices like the DMF32 do DMA output.
      C5 DD 0EF3 2580
      54 DD 0EF3 2581 BISW2 #TTY$M_PC_DMAENA!TTY$M_PC_PRMMAP!TTY$M_PC_NOTIME,-
      08 B2 16 0EF5 2582 UCB$W_TT_PRTCTL(R5)
      16 0EF8 2583 PUSHL R4
      0EFA 2584 JSB @PORT_SET_LINE(R2) ; Jump to tell the port
      0EFD 2585
      0EFD 2586 ; Now call the port to clear the line.
      0EFD 2587
      20 B2 16 0EFD 2588 JSB @PORT_ABORT(R2)
      24 B2 16 0F00 2589 JSB @PORT_RESUME(R2)
      54 8ED0 0F03 2590 POPL R4
      0F06 2591 ENBINT
      44 A5 8ED0 0F09 2592 POPL UCB$L_DEVDEPEND(R5) ; Restore the devdepend field
      0F0D 2593
      0F0D 2594
      0F0D 2595
      0F0D 2596 ; Set up DDCMP defaults and other information necessary for the protocol
      0F0D 2597
      03C0 8F BB 0F0D 2598 PUSHR #^M<R6,R7,R8,R9>
      50 0C A4 D0 0F11 2599 MOVL NOB$A_PRO_BUFFER(R4),R0 ; Get protocol buffer
      01D1 C0 0B A5 90 0F15 2600 MOVUB UCB$B_FIPL(R5),GFSB_FIPL+TF$K_LENGTH(R0) ; Set fork IPL for DDCMP
      01D2 C0 5E A5 90 0F1B 2601 MOVUB UCB$B_DIPL(R5),GFSB_DIPL+TF$K_LENGTH(R0) ; Set device IPL for DDCMP
      01C8 C0 94 0F21 2602 CLRB GFSB_STATE+TF$K_LENGTH(R0) ; Make sure state is halted
      4C A0 01C8 C0 DE 0F25 2603 MOVAL TF$K_LENGTH(R0),TF$A_GFB(R0) ; Set address of global field
      38 BB 0F2B 2604 PUSHR #^M<R3,R4,R5>
      58 60 A4 9E 0F2D 2605 MOVAB NOB$Z_DDCMP(R4),R8 ; Set addr of buf with param
      59 78 A4 9E 0F31 2606 MOVAB NOB$Z_DLA_ADDR(R4),R9 ; Set addr of buffer for addresses
      51 34 A4 9E 0F35 2607 MOVAB NOB$L_XMT_INPR(R4),R1 ; Set up addresses
      69 51 D0 0F39 2608 MOVL R1,DLA$A_XMT_INPR(R9)
      51 28 A4 9E 0F3C 2609 MOVAB NOB$Q_POST(R4),R1
      04 A9 51 D0 0F40 2610 MOVL R1,DLA$A_POSTQ(R9)
      55 50 D0 0F44 2611 MOVL R0,R5 ; Get addr of start of TFB
      56 05 9A 0F47 2612 MOVZBL #DLK$C_CHAR,R6 ; Set DDCMP default char
      57 09F0 8F A8 0F4A 2613 CLRL R7 ; Zero the register
      0F4C 2614 BISW #<DLK$M_MSGCNT!- ; indicate chara to set
      0F51 2615 DLK$M_SELTIM!-
      0F51 2616 DLK$M_SELWAIT!-
      0F51 2617 DLK$M_REPTIM!-
      0F51 2618 DLK$M_REPWAIT!-
      0F51 2619 DLK$M_SETDEF>,R7
      FOAC' 30 0F51 2620 BSBW DDCMP
      38 BA 0F54 2621 POPR #^M<R3,R4,R5>
      0F56 2622
      0F56 2623 ; Before starting up the DDCMP timer make sure that the driver can not
      0F56 2624 ; be reloaded or unloaded while the timer is running.
      0F56 2625
      0000000D'EF 04 88 0F56 2626 BISB #DPT$M_NOUNLOAD,DPT$TAB+DPT$B_FLAGS
      0F5D 2627
      0F5D 2628 ; Start up the DDCMP timer. First set up registers with params for call
      0F5D 2629 ; then make the call to the protocol.
      0F5D 2630
      57 56 38 BB 0F5D 2631 PUSHR #^M<R3,R4,R5>
      164C' CF 9A 0F5F 2632 MOVZBL #DLK$C_START_TIMER,R6 ; Set up R6 with the DDCMP comm
      58 55 D0 0F62 2633 MOVAL W^DEVTIMER,R7 ; R7 must have device timer addr
      59 D4 0F67 2634 MOVL R5,R8 ; R8 must have UCB addr
      0F6A 2635 CLRL R9 ; Clear R9
```

55	OC A4	DO	OF6C	2636	MOVL	NOB\$A_PRO_BUFFER(R4),R5	; Get addr of start of TFB
	F08D	30	OF70	2637	BSBW	DDCMP	; Call protocol
	38	BA	OF73	2638	POPR	#^M<R3,R4,R5>	
03C0	8F	BA	OF75	2639	POPR	#^M<R6,R7,R8,R9>	
50	01	3C	OF79	2640	MOVZWL	S^#SS\$_NORMAL,R0	; Set success
		05	OF7C	2641	RSB		
			OF7D	2642			
			OF7D	2643	START_CTRL_ERROR:		
50	DD	OF7D	2644	PUSHL	R0	; Save abort reason	
08CC	30	OF7F	2645	BSBW	SHUTDOWN_LINE	; Shut down the device	
50	8ED0	OF82	2646	POPL	R0	; Restore abort reason	
		05	OF85	2647	RSB		
			OF86	2648			


```
OF86 2650
OF86 2651 :++
OF86 2652 : START_CIRCUIT- Start device and protocol
OF86 2653 :
OF86 2654 : Functional description:
OF86 2655 :
OF86 2656 : This routine sets up the protocol and device characteristics; sets up receive
OF86 2657 : buffers for the device; starts the protocol; and finally gives the first
OF86 2658 : transmit and receives to the board.
OF86 2659 :
OF86 2660 : If a failure occurs the circuit shutdown sequence is entered.
OF86 2661 :
OF86 2662 : INPUTS:
OF86 2663 : R3 = I/O packet
OF86 2664 : R5 = UCB address
OF86 2665 :
OF86 2666 : IMPLICIT INPUTS:
OF86 2667 :
OF86 2668 : IRP$L_MEDIA contains a copy of the mode buffer specified
OF86 2669 : by the user.
OF86 2670 :
OF86 2671 : OUTPUTS:
OF86 2672 : The request is completed.
OF86 2673 :
OF86 2674 :--
OF86 2675 : START_CIRCUIT:
OF86 2676 : BBC #XMSV_STS_ACTIVE,- ; Start the circuit
OF88 2677 : UCB$L_DEVDEPEND(R5),5$ ; If BC then device and
OF88 2678 : MOVZWL #SS$_DEVACTIVE,R0 ; protocol are not active
OF90 2679 : RSB ; Return error
OF91 2680 :
OF91 2681 5$: MOVL UCB$L_NO_BUFFER(R5),R4 ; Get NOB address
OF96 2682 : BBS #NO_DS_V_INITED,- ; If BS then line started
OF98 2683 : UCB$W_DEVSTS(R5),8$
OF98 2684 : MOVZWL #SS$_DEVINACT,R0 ; Else return device inactive
OFA0 2685 : RSB
OFA1 2686 :
OFA1 2687 : Reset bit fields
OFA1 2688 :
OFA1 2689 8$: PUSHR #^M<R6,R7,R8,R9>
OFA5 2690 : BICW #^C<NO_DS_M_INITED>,UCB$W_DEVSTS(R5)
OFA5 2691 : BICL #NOSC_CIR_PAR,UCB$L_DEVDEPEND(R5) ; Reset all Read/Write flags
OFB3 2692 : BSBW SET_CHAR ; Set protocol char
OFB6 2693 : DSBINT #IPC$ POWER ; Interlock powerfail
OFBC 2694 : BBC #UCB$V_POWER,UCB$W_STS(R5),10$ ; BR if no power failed
OFC1 2695 : ENBINT ; Enable interrupts at FIPL
OFC4 2696 : BRB START_POWERFAIL ; Branch to report powerfail
OFC6 2697 10$: BISW #XMSM_STS_ACTIVE,-
OFC6 2698 : UCB$L_DEVDEPEND(R5)
OFC6 2699 : ENBINT ; Set active state
OFCF 2700 : MOVZBL #DLK$C_USRINT,R6 ; Return to fork IPL
OFD2 2701 : BBC #XMSV_CHR_MOP,- ; Set to start the protocol
OFD4 2702 : UCB$L_DEVDEPEND(R5),15$ ; If BC then don't enter
OFD7 2703 : MOVZBL #DLK$M_MAINT,R7 ; maint mode
OFDA 2704 : BRW 20$ ; Set to start maint mode
OFDD 2705 15$: MOVZBL #DLK$M_START,R7 ; Else simply start the prot
OFE0 2706 20$: PUSHR #^M<R3,R4,R5> ; Save registers
```



```
55 0C A4 D0 OFE2 2707      MOVL      NOBSA_PRO_BUFFER(R4),R5      ; Get addr of start of TFB
    58 7C OFE6 2708      CLRQ      R8
    F015' 30 OFE8 2709      BSBW      DDCMP
    38 BA OFEB 2710      POPR      #^M<R3,R4,R5>
56 09 91 OFED 2711      CMPB      #DLK$C_ACTNOTCOM,R6      ; Jump to the protocol
    13 13 OFF0 2712      BEQL      25$      ; Restore registers
    007B 30 OFF2 2713      BSBW      FILLFREELIST      ; If EQL then protocol not
1E 50 E9 OFF5 2714      BLBC      R0,START_CIRCUIT_ERROR      ; in halted state don't reinit
    51 D4 OFF8 2715      CLRL      R1      ; Fill rcv free buffer Q
    FC4A 30 OFFA 2716      BSBW      START_TRANSMIT      ; Branch to shutdown the device
16 50 E9 OFFD 2717      BLBC      R0,START_CIRCUIT_ERROR      ; Set no status for start transmit
50 01 3C 1000 2718      MOVZWL     S^#SS$_NORMAL,R0      ; Branch to send STRT message
    19 11 1003 2719      BRB       START_CIRCUIT_COMP      ; Branch to shutdown the device
    1005 2720      ; Set normal return
50 02C4 8F 3C 1005 2721 25$: MOVZWL #SS$ _DEACTIVE,R0      ; Comp the request
    100A 2722      SETBIT      #XMS$V_ERR_TRIB,-
    100A 2723      UCB$$_DEVDEPEND(R5)
    05 11 100F 2724      BRB       START_CIRCUIT_ERROR
1011 2725 START_POWERFAIL:
50 0364 8F 3C 1011 2726      MOVZWL #SS$ _POWERFAIL,R0      ; Set reason for abort
    1016 2727 START_CIRCUIT_ERROR:
    50 DD 1016 2728      PUSHL     R0      ; Set to restart circuit
    08C5 30 1018 2729      BSBW      SHUTDOWN_CIRCUIT
    50 8ED0 101B 2730      POPL      R0
    101E 2731 START_CIRCUIT_COMP:
53 58 A5 D0 101E 2732      MOVL      UCB$$_IRP(R5),R3      ; Save reason to abort
7C A5 50 B0 1022 2733      MOVW      R0,UCB$$_BOFF(R5)      ; Shutdown the circuit
    03C0 8F BA 1026 2734      POPR      #^M<R6,R7,R8,R9>      ; Restore reason for abort
    05 102A 2735      RSB
    102B 2736
    102B 2737      ; Retreive packet address
    ; Set success
    ; Jump to complete the request
```



```
102B 2739 .SBTTL Set protocol characteristic
102B 2740 :++
102B 2741 : SET_CHAR - Set characteristics of protocol
102B 2742 :
102B 2743 : FUNCTIONAL DESCRIPTION:
102B 2744 :
102B 2745 : This routine is entered to set characteristic on a halted protocol.
102B 2746 :
102B 2747 : INPUTS:
102B 2748 : R4 = MOB address
102B 2749 : R5 = UCB address
102B 2750 :
102B 2751 :
102B 2752 : OUTPUTS:
102B 2753 : New protocol characteristics given to protocol.
102B 2754 :
102B 2755 :--
102B 2756 SET_CHAR:
58 57 D4 102B 2757 CLRL R7 ; Set no charac to change
60 A4 9E 102D 2758 MOVAB NOB$Z_DDCMP(R4),R8 ; Get trib parameter block
01 A8 90 1031 2759 MOVAB DLK$B_MSGCNT(R8),- ; Use this field for RTO's
02 A8 1034 2760 DLK$B_MAXREP(R8),-
01 A8 90 1036 2761 MOVAB DLK$B_MSGCNT(R8),- ; And for number of Select
03 A8 1039 2762 DLK$B_MAXSEL(R8),- ; intervals
04 A8 B0 103B 2763 MOVW DLK$W_REPWAIT(R8),- ; Use repwait for selwait
06 A8 103E 2764 DLK$W_SELWAIT(R8),-
57 01F8 8F A8 1040 2765 BISW #<DLK$M_MSGCNT!- ; Set to reset these param
1045 2766 DLK$M_SELTIM!-
1045 2767 DLK$M_SELWAIT!-
1045 2768 DLK$M_REPTIM!-
1045 2769 DLK$M_REPWAIT!-
1045 2770 DLK$M_LNTYP>,R7
1045 2771
1045 2772 ASSUME NMA$C_LINPR_POI EQ 0
1045 2773 SETBIT #DLK$V_SETDEF,R7 ; Set default char
70 A4 95 1049 2774 TSTB NOB$B_PRO(R4) ; If EQL then point to point
08 13 104C 2775 BEQL 10$
104E 2776 CLRBIT #DLK$V_SETDEF,R7 ; Use user given char
1052 2777 SETBIT #DLK$V_STATYP,R7 ; Else set trib
1056 2778 10$: ASSUME NMA$C_DPX_FUL EQ 0
71 A4 95 1056 2779 TSTB NOB$B_DUP(R4) ; If EQL the full duplex
04 13 1059 2780 BEQL 20$
105B 2781 SETBIT #DLK$V_DUPLEX,R7 ; Else set half duplex
55 38 BB 105F 2782 20$: PUSHR #^M<R3,R4,R5> ; Save registers
0C A4 D0 1061 2783 MOVL NOB$A_PRO_BUFFER(R4),R5 ; Get addr of start of TFB
56 05 9A 1065 2784 MOVZBL #DLK$C_CHAR,R6 ; Set charac in DDCMP
59 D4 1068 2785 CLRL R9
EF93' 30 106A 2786 BSBW DDCMP
38 BA 106D 2787 POPR #^M<R3,R4,R5> ; Restore the registers
05 05 106F 2788 30$: RSB
1070 2789
1070 2790
```



```
1070 2792 .SBTTL FILLFREELIST - FILL MESSAGE FREE LIST
1070 2793 :++
1070 2794 : FILLFREELIST - Fill message block free list
1070 2795 : ADDFREELIST - Add a buffer to receive list
1070 2796 :
1070 2797 : Functional description:
1070 2798 :
1070 2799 : This routine fills the receive buffer free list up to the quota specified
1070 2800 : at device startup.
1070 2801 :
1070 2802 : Inputs:
1070 2803 :
1070 2804 : R2 = Buffer address (ADDFREELIST only)
1070 2805 : R4 = NOB address
1070 2806 : R5 = UCB address
1070 2807 :
1070 2808 : IPL = FIPL
1070 2809 :
1070 2810 : Outputs:
1070 2811 : R3,R4,R5 are preserved
1070 2812 : R1,R2 destroyed.
1070 2813 :--
1070 2814 FILLFREELIST:
1070 2815 CLRL R2 : Fill free list
1072 2816 BBS #XMSV_STS_ACTIVE,- : Clear buffer address
1074 2817 UCB$W_DEVDEPEND(R5),ADDFREELIST : Continue if device active
1077 2818 RSB
1078 2819 ADDFREELIST: : Add to receive buffer list
1078 2820 PUSHL R3 : Save registers
107A 2821 5$: CMPW UCB$W_DEVBUFSIZ(R5),- : Can new block be allocated ?
107D 2822 NOB$W_QUOTA(R4)
107F 2823 BGTRU 20$ : Br if no - list filled
1081 2824 CLRL R1 : Zero size
1083 2825 ADDW3 #RCV T DATA+CXB$C TRAILER,- : Compute needed block size
1087 2826 UCB$W_DEVBUFSIZ(R5),R1
108A 2827 TSTL R2 : Buffer allocated already?
108C 2828 BNEQ 7$ : Br if yes
108E 2829 JSB G^EX$ALONONPAGED : Allocate nonpaged memory
1094 2830 BLBC R0,10$ : Br if failure
1097 2831 7$: MOVW R1,RCV W_BLKSIZE(R2) : Insert block size
109B 2832 MOVW S^#DYN$C-NET,RCV B_BLKTYPE(R2) : Insert block type
109F 2833 CLRW RCV W_ERROR(R2) : Clear error status
10A2 2834 INSQUE (R2),NOB$Q_FREE(R4) : Insert block on list
10A6 2835 SUBW UCB$W_DEVBUFSIZ(R5),- : Decrement quota
10A9 2836 NOB$W_QUOTA(R4)
10AB 2837 CLRL R2 : Clear buffer pointer
10AD 2838 BRB 5$
10AF 2839
10AF 2840 10$: SETBIT #XMSV_STS_BUFFAIL,- : Set buffer alloc failure
10AF 2841 UCB$W_DEVDEPEND(R5)
10B4 2842 BRB 30$
10B6 2843
10B6 2844 20$: CLRBIT #XMSV_STS_BUFFAIL,- : Clear buffer alloc failure
10B6 2845 UCB$W_DEVDEPEND(R5)
10BB 2846 MOVL R2,R0 : Set address of buffer
10BE 2847 BEQL 30$ : Br if none
10C0 2848 JSB G^COM$DRVDEALMEM : Deallocate it
```


NODRIVER
V04-001

- VAX/VMS DMF32 Async DDMCP Line Driver N 1
FILLFREELIST - FILL MESSAGE FREE LIST 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00
6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 Page 67
(47)

50	01	3C	10C6	2849	30\$:	MOVZWL	S^#SS\$ _NORMAL,R0	; Set for normal return
	53	8ED0	10C9	2850		POPL	R3	; Restore registers
		05	10CC	2851		RSB		
			10CD	2852				


```
10CD 2854 .SBTTL START_RECEIVE and GET_RECEIVE - Start receive and Get a receive buff
10CD 2855 :++
10CD 2856 : START_RECEIVE - Start receives
10CD 2857 :
10CD 2858 : FUNCTIONAL DESCRIPTION:
10CD 2859 :
10CD 2860 : This routine sets up the driver to start receive the header. The header is
10CD 2861 : receive into some fields of the NOB so that even if no buffer has been
10CD 2862 : allocated or can be allocated we can still get the header and the byte
10CD 2863 : count of the message.
10CD 2864 :
10CD 2865 : INPUTS:
10CD 2866 : R4 = NOB address
10CD 2867 : R5 = UCB address
10CD 2868 : IPL = Device IPL
10CD 2869 :
10CD 2870 : OUTPUTS:
10CD 2871 : R4, R5 are preserved
10CD 2872 :
10CD 2873 :--
10CD 2874 : START_RECEIVE:
43 A4 94 10CD 2875 : CLRB NOB$B_RSTATE(R4) ; Reset receiver state
10D0 2876 :
10D0 2877 : ASSUME NOB$W_FLAGS EQ NOB$W_ERROR+2
10D0 2878 :
52 A4 B4 10D0 2879 : CLRW NOB$W_INDEX(R4) ; Clear index into rcv buffer
54 A4 D4 10D3 2880 : CLRL NOB$W_ERROR(R4) ; Reset flag and error fields
10D6 2881 :
10D6 2882 : Set up message count with size of header and crc
10D6 2883 :
50 A4 08 9B 10D6 2884 : MOVZBW #NOB$C_HEADER_HCRC,NOB$W_MSGSIZ(R4)
05 10DA 2885 : RSB
10DB 2886 :
10DB 2887 :
10DB 2888 :++
10DB 2889 : GET_RECEIVE - Get a receive buffer if one is available
10DB 2890 :
10DB 2891 : FUNCTIONAL DESCRIPTION:
10DB 2892 :
10DB 2893 : This routine sets up the driver to receive the data. It gets the next
10DB 2894 : free buffer from the queue. And sets up the fields in the
10DB 2895 : buffer to start receive the data. This buffer is posted
10DB 2896 : when the driver has finished receiveing the data for it.
10DB 2897 :
10DB 2898 : INPUTS:
10DB 2899 : R4 = NOB address
10DB 2900 : R5 = UCB address
10DB 2901 : IPL = Device IPL
10DB 2902 :
10DB 2903 : OUTPUTS:
10DB 2904 : R0 = LBC no buffer available
10DB 2905 : LBS buffer is available and returned in R2
10DB 2906 : R2 = Address of buffer to receive into if a buffer was available
10DB 2907 : R4, R5 are preserved
10DB 2908 :
10DB 2909 :--
10DB 2910 : GET_RECEIVE:
```



```

50 D4 10DB 2911 CLRL R0 ; Assume no buffer
10DD 2912
10DD 2913 ; If line is in process of being shutdown do not try to get a receive buffer
10DD 2914
17 68 A5 0A E0 10DD 2915 BBS #NO_DS_V_SHUTDOWN,UCB$W_DEVSTS(R5),10$
52 20 B4 0F 10E2 2916 REMQUE @NOB$Q_FREE(R4),R2 ; Get next available buffer
11 1D 10E6 2917 BVS 10$ ; If VS none available
30 A4 52 D0 10E8 2918 MOVL R2,NOB$W_L_RCV_INPR(R4) ; Set receive buffer address
10EC 2919
10EC 2920 ASSUME NOB$W_INDEX EQ NOB$W_MSGSIZ+2
10EC 2921 ASSUME NOB$W_ERROR EQ NOB$W_INDEX+2
10EC 2922 ASSUME NOB$W_FLAGS EQ NOB$W_ERROR+2
10EC 2923
10EC 2924 ASSUME RCV_W_INDEX EQ RCV_W_MSGSIZ+2
10EC 2925 ASSUME RCV_W_ERROR EQ RCV_W_INDEX+2
10EC 2926 ASSUME RCV_W_FLAGS EQ RCV_W_ERROR+2
10EC 2927
0C A2 50 A4 7D 10EC 2928 MOVQ NOB$W_MSGSIZ(R4),RCV_W_MSGSIZ(R2)
10F1 2929
10F1 2930 ASSUME NOB$C_HEADER_HCRC EQ 8
10F1 2931
40 A2 48 A4 7D 10F1 2932 MOVQ NOB$Z_HEADER(R4),RCV_Z_HEADER(R2)
50 01 3C 10F6 2933 MOVZWL S^#SS$_NORMAL,R0 ; Set normal return
05 10F9 2934 10$: RSB
10FA 2935
```



```
10FA 2937 .SBTTL NOSGETNXT - Class driver GETNXT routine
10FA 2938 :++
10FA 2939 : NOSGETNXT - Class driver GETNXT routine entry point for PORT driver
10FA 2940 :
10FA 2941 : FUNCTIONAL DESCRIPTION
10FA 2942 :
10FA 2943 : This is the ASYNCH DDCMP transmit "interrupt" service routine. This routine
10FA 2944 : will be called whenever the terminal port driver has completed the current
10FA 2945 : character or burst of characters given to it by the ASYNCH DDCMP class driver.
10FA 2946 : When called this routine will calculate the next character or burst of
10FA 2947 : characters to transmit over the port driver. If data is returned by
10FA 2948 : GETNXT then a timer is set in the port driver and the bit UCB$V_INT
10FA 2949 : must be set in UCB$W_STS to say that the port is still busy. Remember that
10FA 2950 : messages are transmitted over the port driver in chunks of data (See
10FA 2951 : START_TRANSMIT to get break down).
10FA 2952 :
10FA 2953 : Note that UCB$B_IT_OUTTYPE is always set to negative (unless there is
10FA 2954 : no data to send), because we always send data in bursts.
10FA 2955 :
10FA 2956 : INPUTS:
10FA 2957 :     R5 = UCB address
10FA 2958 :
10FA 2959 : IPL = DIPL
10FA 2960 :
10FA 2961 : OUTPUTS:
10FA 2962 :     R3 = character to output
10FA 2963 :     R5 = UCB address
10FA 2964 :
10FA 2965 : IMPLICIT OUTPUTS:
10FA 2966 :     UCB$B_IT_OUTTYPE = Zero - no data to output
10FA 2967 :                       One - single character in R3
10FA 2968 :                       Negative - output a burst of data
10FA 2969 :
10FA 2970 :     Burst mode only:
10FA 2971 :     UCB$L_IT_OUTADR = Address of the first character of the burst
10FA 2972 :     UCB$W_IT_OUTLEN = Length of the burst
10FA 2973 :
10FA 2974 :     R0 preserved, All others destroyed
10FA 2975 : --
10FA 2976 : NOSGETNXT:
10FA 2977 :     PUSHR #^M<R0,R4,R6,R7,R8,R9>
10FA 2978 :     MOVL  UCB$L_NO_BUFFER(R5),R4 ; Get NOB address
10FA 2979 :     BEQL  20$
10FA 2980 :     MOVL  NOB$L_XMT_INPR(R4),R8 ; Get transmit address
10FA 2981 :     BEQL  30$ ; If EQL no buffer to transmit
10FA 2982 :     SETBIT #UCB$V_INT,UCB$W_STS(R5) ; Assume we always have data to send
10FA 2983 :
10FA 2984 : ; Increment the transmitter state to say that we should send the next block
10FA 2985 : ; of the transmit. If we increment past the sending of pads then that
10FA 2986 : ; means we are thru with this data block and we should post this
10FA 2987 : ; data message for complete and start the next message transmitting.
10FA 2988 :     ASSUME NOSC_HEADER EQ NOSC_EMPTY+1
10FA 2989 :     ASSUME NOSC_HCRC EQ NOSC_HEADER+1
10FA 2990 :     ASSUME NOSC_DATA EQ NOSC_HCRC+1
10FA 2991 :     ASSUME NOSC_DCRC EQ NOSC_DATA+1
10FA 2992 :     ASSUME NOSC_PADS EQ NOSC_DCRC+1
10FA 2993 :
```

54	03D1	8F	BB	10FA	2977			
	00A8	C5	D0	10FE	2978			
		57	13	1103	2979			
58	34	A4	D0	1105	2980			
		63	13	1109	2981			
				110B	2982			
				1110	2983			
				1110	2984			
				1110	2985			
				1110	2986			
				1110	2987			
				1110	2988			
				1110	2989			
				1110	2990			
				1110	2991			
				1110	2992			
				1110	2993			


```

42 A4 0A 91 1110 2994 10$: CMPB #NOSC_XMTERR,NOBSB_XSTATE(R4) ; If NEQ normal xmt states
03 12 1114 2995 BNEQ 12$
013C 31 1116 2996 BRW SEND_XMTERR ; Else handle the xmt error
42 A4 96 1119 2997 12$: INCB NOBSB_XSTATE(R4)
50 42 A4 9A 111C 2998 MOVZBL NOBSB_XSTATE(R4),R0 ; Get state for dispatch
1120 2999 $DISPATCH R0,TYPE=B,-
1120 3000 <- ; Function ; Action
1120 3001 <NOSC_EMPTY 25$>,-
1120 3002 <NOSC_HEADER SEND_HEADER>,-
1120 3003 <NOSC_HCRC SEND_HCRC>,-
1120 3004 <NOSC_DATA SEND_DATA>,-
1120 3005 <NOSC_DCRC SEND_DCRC>,-
1120 3006 <NOSC_PADS SEND_PADS>,-
1120 3007 >
1130 3008
1130 3009 ; Fall thru to complete the buffer and start up the next transfer if possible
1130 3010
1130 3011 14$: CLRBIT #NO_DS_V_XMT_TIME,UCBSW_DEVSTS(R5) ; Clear due time not valid
OC 1F A8 02 E0 1135 3012 CLRL NOBSL_XMT_INPR(R4) ; Clear addr from inprogress
2C B4 68 0E 1138 3013 BBS #XMTQSV_CONTROL,XMTQSB_FLAG(R8),15$ ; If BS cntrl msg don't post
54 DD 1141 3014 INSQUE (R8),@NOBSQ_POST+4(R4) ; Post the buffer for completion
0272 30 1143 3015 PUSHL R4
54 8ED0 1146 3016 BSBW SCHED_FORK_IO ; Schedule the fork
42 A4 94 1149 3017 POPL R4
OB 68 A5 04 E0 1149 3018 15$: CLRB NOBSB_XSTATE(R4) ; Reset the transmitter state
114C 3019 BBS #NO_DS_V_XMTING,UCBSW_DEVSTS(R5),20$ ; If BS then turn link don't xm
1151 3020
1151 3021 ; If the following bit is set then the circuit is going thru shutdown,
1151 3022 ; therefore do not start up any other transmits over the port.
1151 3023
06 68 A5 0A E0 1151 3024 BBS #NO_DS_V_SHUTDOWN,UCBSW_DEVSTS(R5),20$
FB86 30 1156 3025 BSBW GET_XMT ; Else get next transmit to send
B4 50 E8 1159 3026 BLBS R0,T0$ ; If LBS then data to send
03D1 8F BA 115C 3027 20$: CLRBIT #UCBSV_INT,UCBSW_STS(R5) ; Set port not busy
1161 3028 POPR #^M<R0,R4,R6,R7,R8,R9> ; Restore registers
1165 3029
1165 3030 ; The following instruction must be the last instruction executed before
1165 3031 ; the return to the port input routine. The reason for this is that the
1165 3032 ; condition codes must correctly so that the port does the right transfer
1165 3033
010B C5 94 1165 3034 CLRB UCBSB_TT_OUTTYPE(R5) ; Else set no data to send
05 1169 3035 RSB
116A 3036
116A 3037 25$: BUG_CHECK NOBUFCKT,FATAL
116E 3038
116E 3039 ; This code is here because DDCMP retransmission of messages must take the
116E 3040 ; message in progress away from the device. This is to ensure that
116E 3041 ; messages are not transmitted out of order. Since we do not want to
116E 3042 ; cut the message short we will send pads until the message byte count
116E 3043 ; has run out.
116E 3044
42 A4 95 116E 3045 30$: TSTB NOBSB_XSTATE(R4) ; If EQL not sending anything
E9 13 1171 3046 BEQL 20$
50 42 A4 90 1173 3047 MOVB NOBSB_XSTATE(R4),R0 ; Get state
50 50 96 1177 3048 INCB R0 ; Set to next state
42 A4 0A 90 1179 3049 MOVB #NOSC_XMTERR,NOBSB_XSTATE(R4) ; Set new state for xmt
117D 3050 $DISPATCH R0,TYPE=B,-

```



```
117D 3051      <-      :Function      : Action
117D 3052      <NOSC_EMPTY      25$>,-
117D 3053      <NOSC_HEADER      40$>,-
117D 3054      <NOSC_HCRC      45$>,-
117D 3055      <NOSC_DATA      50$>,-
117D 3056      <NOSC_DCRC      55$>,-
117D 3057      <NOSC_PADS      60$>,-
117D 3058      >
50 46 A4 B0 118D 3059      MOVW      NOBSW_XMTERR_SIZE(R4),R0
11 1191 3060      BRB      65$
1193 3061
50 42 A5 0A A1 1193 3062 40$:      ADDW3      #10,UCBSW_DEVBUSIZ(R5),R0      ; Set the bytes to xmt given state
18 11 1198 3063      BRB      65$
50 42 A5 04 A1 119A 3064 45$:      ADDW3      #4,UCBSW_DEVBUSIZ(R5),R0
11 11 119F 3065      BRB      65$
50 04 9B 11A1 3066 50$:      MOVZBW      #4,R0
0C 11 11A4 3067      BRB      65$
50 02 9B 11A6 3068 55$:      MOVZBW      #2,R0
07 11 11A9 3069      BRB      65$
50 02 9B 11AB 3070 60$:      MOVZBW      #2,R0
42 A5 05 90 11AE 3071      MOVB      #NOSC_PADS,NOBSB_XSTATE(R5)
34 A4 0080 C4 DE 11B2 3072 65$:      MOVAL      NOBSZ_CTL_MSG(R4),NOBSL_XMT_INPR(R4) ; Set to fake out xmt
46 A4 50 B0 11B8 3073      MOVW      R0,NOBSW_XMTERR_SIZE(R4)
011C C5 44 A4 DE 11BC 3074      MOVAL      NOBSW_PADS(R4),UCBSL_TT_OUTADR(R5)
0120 C5 02 9B 11C2 3075      MOVZBW      #2,UCBSW_TT_OUTLEN(R5)
010B C5 01 8E 11C7 3076      MNEGB      #1,UCBSB_TT_OUTTYPE(R5)
03D1 8F BA 11CC 3077      POPR      #^M<R0,R4,R6,R7,R8,R9>
05 11D0 3078      RSB
11D1 3079
11D1 3080      ; Set up the port to transmit the header portion of the message
11D1 3081
11D1 3082      SEND_HEADER:
42 A4 01 90 11D1 3083      MOVB      #NOSC_HEADER,NOBSB_XSTATE(R4) ; Set sending header state
011C C5 24 A8 DE 11D5 3084      MOVAL      XMTQSB_MSGHDR(R8),UCBSL_TT_OUTADR(R5) ; Set address of header
0120 C5 06 9B 11DB 3085      MOVZBW      #NOSC_HEADER_LEN,UCBSW_TT_OUTLEN(R5) ; Set length of header
03D1 8F BA 11E0 3086      POPR      #^M<R0,R4,R6,R7,R8,R9>
11E4 3087
11E4 3088      ; The following instruction must be the last instruction executed before
11E4 3089      ; the return to the port output routine. The reason for this is that the
11E4 3090      ; condition codes must correctly so that the port does the right transfer
11E4 3091
010B C5 01 8E 11E4 3092      MNEGB      #1,UCBSB_TT_OUTTYPE(R5) ; Set block transfer to port
05 11E9 3093      RSB
11EA 3094
11EA 3095      ; Set up the port to transmit the header CRC portion of the message
11EA 3096
11EA 3097      SEND_HCRC:
42 A4 02 90 11EA 3098      MOVB      #NOSC_HCRC,NOBSB_XSTATE(R4) ; Set sending HCRC state
011C C5 20 A8 DE 11EE 3099      MOVAL      XMTQSB_HCRC(R8),UCBSL_TT_OUTADR(R5) ; Set address of HCRC
0120 C5 02 9B 11F4 3100      MOVZBW      #2,UCBSW_TT_OUTLEN(R5) ; Set length of HCRC
03D1 8F BA 11F9 3101      POPR      #^M<R0,R4,R6,R7,R8,R9>
11FD 3102
11FD 3103      ; The following instruction must be the last instruction executed before
11FD 3104      ; the return to the port output routine. The reason for this is that the
11FD 3105      ; condition codes must correctly so that the port does the right transfer
11FD 3106
010B C5 01 8E 11FD 3107      MNEGB      #1,UCBSB_TT_OUTTYPE(R5) ; Set block transfer to port
```



```
05 1202 3108 RSB
1203 3109
1203 3110 ; Set up the port to transmit the data portion of the message
1203 3111
1203 3112 SEND_DATA:
1203 3113 ; If the control bit is set then this is a control message and there is no
1203 3114 ; data to send with this message. Thus we want to send the pads out
1203 3115 ; now.
1203 3116
1203 3117 BBS #XMTQ$V_CONTROL,XMTQ$B_FLAG(R8),SEND_PADS
011C 34 1F A8 02 E0 1203 3117 BBS #XMTQ$V_CONTROL,XMTQ$B_FLAG(R8),SEND_PADS
42 A4 03 90 1208 3118 MOVB #NOSC_DATA,NOB$B_XSTATE(R4) ; Set sending data state
011C 011C C5 2A A8 DE 120C 3119 MOVAL XMTQ$R_LENGTH(R8),UCB$L_TT_OUTADR(R5) ; Set address of data
1212 3120
1212 3121 ; Set length of the data message. Remember that DDCMP adds the header size
1212 3122 ; into the message size, so we should subtract it before giving the size
1212 3123 ; to the port.
1212 3124
1212 3125 SUBW3 #NOSC_HEADER_LEN,XMTQ$W_MSGSIZE(R8),UCB$W_TT_OUTLEN(R5)
0120 0120 C5 1A A8 06 A3 1212 3125 SUBW3 #NOSC_HEADER_LEN,XMTQ$W_MSGSIZE(R8),UCB$W_TT_OUTLEN(R5)
03D1 03D1 8F BA 1219 3126 POPR #^M<R0,R4,R6,R7,R8,R9>
121D 3127
121D 3128 ; The following instruction must be the last instruction executed before
121D 3129 ; the return to the port output routine. The reason for this is that the
121D 3130 ; condition codes must correctly so that the port does the right transfer
121D 3131
121D 3132 MNEGB #1,UCB$B_TT_OUTTYPE(R5) ; Set block transfer to port
010B 010B C5 01 8E 121D 3132 MNEGB #1,UCB$B_TT_OUTTYPE(R5) ; Set block transfer to port
05 05 1222 3133 RSB
1223 3134
1223 3135 ; Set up the port to transmit the data CRC portion of the message
1223 3136
1223 3137 SEND_DCRC:
1223 3138 MOVB #NOSC_DCRC,NOB$B_XSTATE(R4) ; Set sending data crc state
011C 42 A4 04 90 1223 3138 MOVB #NOSC_DCRC,NOB$B_XSTATE(R4) ; Set sending data crc state
011C 011C C5 22 A8 DE 1227 3139 MOVAL XMTQ$W_DCRC(R8),UCB$L_TT_OUTADR(R5) ; Set address of data crc
0120 0120 C5 02 9B 122D 3140 MOVZBW #2,UCB$W_TT_OUTLEN(R5) ; Set length of data CRC
03D1 03D1 8F BA 1232 3141 POPR #^M<R0,R4,R6,R7,R8,R9>
1236 3142
1236 3143 ; The following instruction must be the last instruction executed before
1236 3144 ; the return to the port output routine. The reason for this is that the
1236 3145 ; condition codes must correctly so that the port does the right transfer
1236 3146
1236 3147 MNEGB #1,UCB$B_TT_OUTTYPE(R5) ; Set block transfer to port
010B 010B C5 01 8E 1236 3147 MNEGB #1,UCB$B_TT_OUTTYPE(R5) ; Set block transfer to port
05 05 1238 3148 RSB
123C 3149
123C 3150 ; Set up the port to transmit the pads for the message
123C 3151
123C 3152 SEND_PADS:
123C 3153 MOVB #NOSC_PADS,NOB$B_XSTATE(R4) ; Set sending pads state
011C 42 A4 05 90 123C 3153 MOVB #NOSC_PADS,NOB$B_XSTATE(R4) ; Set sending pads state
011C 011C C5 44 A4 DE 1240 3154 MOVAL NOB$W_PADS(R4),UCB$L_TT_OUTADR(R5) ; Set address of pads
0120 0120 C5 02 9B 1246 3155 MOVZBW #2,UCB$W_TT_OUTLEN(R5) ; Set length of pads
03D1 03D1 8F BA 1248 3156 POPR #^M<R0,R4,R6,R7,R8,R9>
124F 3157
124F 3158 ; The following instruction must be the last instruction executed before
124F 3159 ; the return to the port output routine. The reason for this is that the
124F 3160 ; condition codes must correctly so that the port does the right transfer
124F 3161
124F 3162 MNEGB #1,UCB$B_TT_OUTTYPE(R5) ; Set block transfer to port
010B 010B C5 01 8E 124F 3162 MNEGB #1,UCB$B_TT_OUTTYPE(R5) ; Set block transfer to port
05 05 1254 3163 RSB
1255 3164
```



```

1255 3165
1255 3166 ; There was an error on the line and this mechanism is used to ensure
1255 3167 ; that the other side gets enough data to cause a CRC error. Please
1255 3168 ; note that that the last time thru we set the PAD state in order
1255 3169 ; to start up the next XMT
1255 3170
1255 3171 SEND_XMTERR:
1255 3172 SUBW2 #2,NOBSW_XMTERR_SIZE(R4) ; Decr number of pads to send
1259 3173 BGTR 10$ ; If gtr continue
125B 3174 MOVB #NOSC_PADS,NOBSB_XSTATE(R4) ; Else reset state to start transmit
011C 0120 C5 44 A4 DE 125F 3175 10$: MOVAL NOBSW_PADS(R4),UCBSL_TT_OUTADR(R5) ; Set address of pads
0120 C5 02 9B 1265 3176 MOVZBW #2,UCBSW_TT_OUTLEN(R5) ; Set length of pads
03D1 8F BA 126A 3177 POPR #^M<R0,R4,R6,R7,R8,R9>
126E 3178
126E 3179 ; The following instruction must be the last instruction executed before
126E 3180 ; the return to the port output routine. The reason for this is that the
126E 3181 ; condition codes must correctly so that the port does the right transfer
126E 3182
010B C5 01 8E 126E 3183 MNEGB #1,UCBSB_TT_OUTTYPE(R5) ; Set block transfer to port
05 1273 3184 RSB

```



```
1274 3186 .SBTTL NOSPUTNXT - Async class driver receive character routine
1274 3187 :++
1274 3188 : NOSPUTNXT - Async class driver receive character routine
1274 3189 :
1274 3190 : FUNCTIONAL DESCRIPTION:
1274 3191 :
1274 3192 : This routine is called whenever the terminal port driver has received a
1274 3193 : character. Because of the nature of ASYNCH DDCMP this routine will
1274 3194 : not return data to transmit over the port.
1274 3195 :
1274 3196 : INPUTS:
1274 3197 : R3 = Input character
1274 3198 : R5 = UCB address
1274 3199 :
1274 3200 : OUTPUTS:
1274 3201 : R5 = UCB address
1274 3202 : R0 is preserved ALL other registers are destroyed
1274 3203 :
1274 3204 : IMPLICIT OUTPUTS:
1274 3205 : UCBSB_TT_OUTTYPE = Zero no data to send
1274 3206 :
1274 3207 :--
1274 3208 NOSPUTNXT:
1274 3209 PUSHF #M<R0,R4,R6,R7,R8,R9>
1278 3210 MOVL UCBSL_NO_BUFFER(R5),R4 ; Get address of NOB
127D 3211 BEQL 30$ ; If EQL device not started
127F 3212 10$: MOVZBL NOBSB_RSTATE(R4),R0 ; Get receiver state
1283 3213 $DISPATCH R0,TYPE=B,-
1283 3214 <- ; Action
1283 3215 <NOSC_SEAR MSG SEARCH MESSAGE>,-
1283 3216 <NOSC_HEADER RCV_HEADER>,-
1283 3217 <NOSC_DATA RCV_DATA>,-
1283 3218 <NOSC_NOBUFFER RCV_NOBUFFER>,-
1283 3219 >
1291 3220 BUG_CHECK NOBUFCKT,FATAL
1295 3221
1295 3222 30$: POPR #M<R0,R4,R6,R7,R8,R9> ; Else ignore the char and return
1299 3223
1299 3224 ; The following instruction must be the last instruction executed before
1299 3225 ; the return to the port input routine. The reason for this is that the
1299 3226 ; condition codes must correctly so that the port does the right transfer
1299 3227
1299 3228 CLR B UCBSB_TT_OUTTYPE(R5) ; Else set no data to send
129D 3229 RSB
129E 3230
129E 3231 ; This searches for the start of a DDCMP message. All characters are ignored
129E 3232 ; until a start of message is received.
129E 3233
129E 3234 SEARCH_MESSAGE:
129E 3235 CMPB R3,#NOSC_SOH ; Is this a start of a data message
12A2 3236 BEQL 10$ ; If EQL then it is
12A4 3237 CMPB R3,#NOSC_DLE ; Start of a maintenance msg?
12A8 3238 BEQL 10$ ; If EQL then yes
12AA 3239 CMPB R3,#NOSC_ENQ ; Start of control message?
12AD 3240 BNEQ 20$ ; If NEQ then no
12AF 3241 SETBIT #RCV_V_CNTL,NOBSW_FLAGS(R4) ; Set this is a control msg
12B4 3242
```

54 03D1 8F BB 1274 3209
00A8 C5 D0 1278 3210
16 13 127D 3211
50 43 A4 9A 127F 3212 10\$:
1283 3213
1283 3214
1283 3215
1283 3216
1283 3217
1283 3218
1283 3219
1291 3220
1295 3221
03D1 8F BA 1295 3222 30\$:
1299 3223
1299 3224
1299 3225
1299 3226
1299 3227
010B C5 94 1299 3228
05 129D 3229
129E 3230
129E 3231
129E 3232
129E 3233
81 8F 53 91 129E 3234
10 13 12A2 3235
90 8F 53 91 12A4 3236
0A 13 12A8 3237
05 53 91 12AA 3238
35 12 12AD 3239
12AF 3240
12B4 3241
12B4 3242


```
12B4 3243 ASSUME NOSC_HEADER EQ NOSC_SEAR_MSG+1
12B4 3244
12B4 3245 ; The terminal line receive speed is store in the UCB$W_TT_SPEED+1 unless
12B4 3246 ; the field is zero in which case it is the transmit speed
50 00F5 C5 9A 12B4 3247 10$: MOVZBL UCB$W_TT_SPEED+1(R5),R0
12B4 3248 BNEQ 15$
50 00F4 C5 9A 12BB 3249 MOVZBL UCB$W_TT_SPEED(R5),R0
50 EE3F CF40 3C 12C0 3250 15$: MOVZWL DUETIM_TABLE[R0],R0
51 00000000'GF D0 12C6 3251 MOVL G^EXESGL ABSTIM,R1
38 A4 51 50 C1 12CD 3252 ADDL3 R0,R1,NOBSL_RCV DUETIM(R4)
12D2 3253 SETBIT #NO_DS_V_RCV TIME,UCB$W_DEVSTS(R5)
43 A4 96 12D7 3254 INCB NOBSB_RSTATE(R4) ; Move to next state rcving hdr
48 A4 53 90 12DA 3255 MOVB R3,NOBSZ_HEADER(R4) ; Save character rcvd in hdr area
52 A4 B6 12DE 3256 INCW NOBSW_INDEX(R4) ; Incr the index into buffer
50 A4 B7 12E1 3257 DECW NOBSW_MSGSIZ(R4) ; Decr the number of bytes to rcv
03D1 8F BA 12E4 3258 20$: POPR #^M<R0,R4,R6,R7,R8,R9>
12E8 3259
12E8 3260 ; The following instruction must be the last instruction executed before
12E8 3261 ; the return to the port input routine. The reason for this is that the
12E8 3262 ; condition codes must correctly so that the port does the right transfer
12E8 3263
010B C5 94 12E8 3264 CLRB UCB$B_TT_OUTYPE(R5) ; Else set no data to send
05 12EC 3265 RSB
12ED 3266
12ED 3267 ; This routine inputs each character into the header of the rcv buffer until
12ED 3268 ; both the header and header CRC are input.
12ED 3269
12ED 3270 RCV_HEADER:
50 52 A4 3C 12ED 3271 MOVZWL NOBSW_INDEX(R4),R0 ; Get index into rcv buffer
51 48 A4 DE 12F1 3272 MOVAL NOBSZ_HEADER(R4),R1 ; Get pointer to start of rcv buffer
6140 53 90 12F5 3273 MOVB R3,(RT)[R0] ; Move char into buffer
52 A4 B6 12F9 3274 INCW NOBSW_INDEX(R4) ; Incr the index
50 A4 B7 12FC 3275 DECW NOBSW_MSGSIZ(R4) ; Decr the bytes to rcv
26 14 12FF 3276 BGTR 10$ ; If GTR still more bytes to rcv
1301 3277
1301 3278 ; After the entire header and crc's are received copy them into a buffer.
1301 3279 ; Set up the unit to make it start receiving the data into the message
1301 3280
1301 3281 ; BBS #NO_DS_V_SHUTDOWN,UCB$W_DEVSTS(R5),30$ ; If device is shut dont star
FDD7 30 1301 3282 BSBW GET_RECEIVE ; Start receive
29 50 E9 1304 3283 BLBC R0,20$ ; If LBS then buffer to rcv into
1307 3284
1307 3285 ASSUME RCV_V_CNTL EQ 0
1307 3286
67 12 A2 E8 1307 3287 BLBS RCV_W_FLAGS(R2),COMP_RCV ; If LBS complete the control msg
50 01 A1 B0 130B 3288 MOVW MFD$W_CNTLFLG(R1),R0 ; Else get size of data portion of m
50 C000 8F AA 130F 3289 BICW2 #<MFD$M_SELECT!MFD$M_QSYNC>,R0 ; Clear the flags from the msg size
1314 3290
1314 3291 ; The following code is here so that we can catch the fact that systems run
1314 3292 ; with different buffer sizes. We don't want to corrupt pool if
1314 3293 ; we receive a message that is too large for our receive buffer. Incidentally,
1314 3294 ; the CRC checking should catch this as an error and drop the message on the
1314 3295 ; floor.
1314 3296
50 42 A5 B1 1314 3297 CMPW UCB$W_DEVBUSIZ(R5),R0 ; If GEQ then not larger than our bu
04 18 1318 3298 BGEQ 5$
50 42 A5 B0 131A 3299 MOVW UCB$W_DEVBUSIZ(R5),R0 ; Use the smaller of the two
```



```
OC A2 50 02 A1 131E 3300 5$: ADDW3 #2,R0,RCV_W_MSGSIZ(R2) ; Set size of msg plus 2 for CRC
43 A4 03 90 1323 3301 MOVW #NOSC_DATA,NOBSB_RSTATE(R4) ; Set receive data state
03D1 8F BA 1327 3302 10$: POPR #^M<R0,R4,R6,R7,R8,R9>
132B 3303
132B 3304 ; The following instruction must be the last instruction executed before
132B 3305 ; the return to the port input routine. The reason for this is that the
132B 3306 ; condition codes must correctly so that the port does the right transfer
132B 3307
010B C5 94 132B 3308 CLR B UCB$B_TT_OUTTYPE(R5) ; Else set no data to send
05 132F 3309 RSB
1330 3310
1330 3311 ; Come here when we did not have a buffer in which to receive the data
1330 3312
4D 56 A4 E8 1330 3313 20$: BLBS NOBSW_FLAGS(R4),COMP_RCV_NOBUFFER ; If LBS complete the control msg
50 01 A1 B0 1334 3314 MOVW MFD$W_CNTFLG(R1),R0 ; Else get size of data portion of m
50 C000 8F AA 1338 3315 BICW2 #<MFD$M_SELECT!MFD$M_QSYNC>,R0 ; Clear the flags from the msg size
50 A4 50 02 A1 133D 3316 ADDW3 #2,R0,NOBSW_MSGSIZ(R4) ; Set size of msg plus 2 for CRC
43 A4 04 90 1342 3317 MOVW #NOSC_NOBUFFER,NOBSB_RSTATE(R4) ; Set nobuffer receive data state
03D1 8F BA 1346 3318 POPR #^M<R0,R4,R6,R7,R8,R9>
134A 3319
134A 3320 ; The following instruction must be the last instruction executed before
134A 3321 ; the return to the port input routine. The reason for this is that the
134A 3322 ; condition codes must correctly so that the port does the right transfer
134A 3323
010B C5 94 134A 3324 CLR B UCB$B_TT_OUTTYPE(R5) ; Else set no data to send
05 134E 3325 RSB
134F 3326
134F 3327 ; This routine inputs each character into the data part of the rcv buffer until
134F 3328 ; both the data and data CRC are input.
134F 3329
52 30 A4 D0 134F 3330 RCV_DATA:
14 13 1353 3331 MOVL NOBSL_RCV_INPR(R4),R2 ; Get buffer
50 0E A2 3C 1355 3332 BEQL 10$ ; Branch to return if not there
51 40 A2 DE 1359 3333 MOVZWL RCV_W_INDEX(R2),R0 ; Get index into rcv buffer
6140 53 90 135D 3334 MOVAL RCV_W_HEADER(R2),R1 ; Get pointer to start of rcv buffer
0E A2 B6 1361 3335 MOVW R3,(RT)[R0] ; Move char into buffer
0C A2 B7 1364 3336 INCW RCV_W_INDEX(R2) ; Incr the index
09 15 1367 3337 DECB RCV_W_MSGSIZ(R2) ; Decr the bytes to rcv
03D1 8F BA 1369 3338 BLEQ COMP_RCV ; If LEQ then rcv is complete
136D 3339 10$: POPR #^M<R0,R4,R6,R7,R8,R9>
136D 3340
136D 3341 ; The following instruction must be the last instruction executed before
136D 3342 ; the return to the port input routine. The reason for this is that the
136D 3343 ; condition codes must correctly so that the port does the right transfer
136D 3344
010B C5 94 136D 3345 CLR B UCB$B_TT_OUTTYPE(R5) ; Else set no data to send
05 1371 3346 RSB
1372 3347
1372 3348 ; This routine posts the current receive for IO completion then sets up
1372 3349 ; up the next buffer into which to receive.
1372 3350
30 A4 D4 1372 3351 COMP_RCV:
2C B4 62 0E 1375 3352 CLRL NOBSL_RCV_INPR(R4) ; Clear no rcv buffer inprg
54 DD 1379 3353 INSQUE (R2),NOBSQ_POST+4(R4) ; Send buffer to post
003A 30 137B 3354 PUSHL R4
54 8ED0 137E 3355 BSBW SCHED_FORK_IO ; Set up fork
137E 3356 POPL R4
```


			1381	3357	COMP_RCV_NOBUFFER:	
			1381	3358	CLRB	#NO DS V RCV TIME,UCB\$W_DEVSTS(R5) ; Set due time not valid
FD44	30		1386	3359	BSBW	START RECEIVE ; Set up next receive
03D1	8F	BA	1389	3360	POPR	#^M<R0,R4,R6,R7,R8,R9>
			138D	3361		
			138D	3362		; The following instruction must be the last instruction executed before
			138D	3363		; the return to the port input routine. The reason for this is that the
			138D	3364		; condition codes must correctly so that the port does the right transfer
			138D	3365		
010B	C5	94	138D	3366	CLRB	UCB\$B_TT_OUTTYPE(R5) ; Else set no data to send
		05	1391	3367	RSB	
			1392	3368		
			1392	3369	RCV_NOBUFFER:	
50	A4	B7	1392	3370	DECW	NOB\$W_MSGSIZ(R4)
	EA	15	1395	3371	BLEQ	COMP_RCV_NOBUFFER
03D1	8F	BA	1397	3372	POPR	#^M<R0,R4,R6,R7,R8,R9>
			139B	3373		
			139B	3374		; The following instruction must be the last instruction executed before
			139B	3375		; the return to the port input routine. The reason for this is that the
			139B	3376		; condition codes must correctly so that the port does the right transfer
			139B	3377		
010B	C5	94	139B	3378	CLRB	UCB\$B_TT_OUTTYPE(R5) ; Else set no data to send
		05	139F	3379	RSB	
			13A0	3380		
			13A0	3381		
			13A0	3382		


```
13A0 3384 .SBTTL NOSCLASS_PORTFORK - Asyncnh DDCMP fork routines
13A0 3385 :++
13A0 3386 : NOSCLASS_PORTFORK - Asyncnh DDCMP port fork routine
13A0 3387 :
13A0 3388 : FUNCTIONAL DESCRIPTION:
13A0 3389 :
13A0 3390 : This routine is called by the port driver when the port driver needs to start
13A0 3391 : up a fork process. This routine only schedules a fork if no fork is already
13A0 3392 : scheduled. If a fork is scheduled then a bit is set in UCB$$_FR3 to indicate
13A0 3393 : that the port driver must be called at its FORK entry point.
13A0 3394 :
13A0 3395 : INPUTS:
13A0 3396 : R5 = UCB address
13A0 3397 :
13A0 3398 : OUTPUTS:
13A0 3399 : FORK may be scheduled
13A0 3400 :
13A0 3401 :--
13A0 3402 NOSCLASS_PORTFORK:
OD 68 A5 07 E2 13A0 3403 BBSS #NO_DS_V_FORK_PEND,UCB$$_DEVSTS(R5),10$ ; If BS then fork pending
1451'CF 3F 13A5 3404 PUSHAW W^FORK_DONE ; Set address of fork routine
53 01 9A 13A9 3405 MOVZBL #NO_FS_M_PORTFORK,R3 ; Set type of fork
00000000'GF 17 13AC 3406 JMP G^EXES$FORK
13B2 3407
13B2 3408 10$: SETBIT #NO_FS_V_PORTFORK,UCB$$_FR3(R5) ; Set port fork pending
05 13B7 3409 RSB
13B8 3410
13B8 3411 :++
13B8 3412 : SCHED_FORK_IO - Schedules forks for asyncnh ddcmp class driver
13B8 3413 :
13B8 3414 : FUNCTIONAL DESCRIPTION:
13B8 3415 :
13B8 3416 : This routine schedules the IO completion forks for the asyncnh class
13B8 3417 : driver. If there is a fork pending on the device then no fork
13B8 3418 : is scheduled but a bit is set in UCB$$_FR3 indicating that there is
13B8 3419 : some IO to complete when the fork is processed.
13B8 3420 :
13B8 3421 : INPUTS:
13B8 3422 : R5 = UCB address
13B8 3423 :
13B8 3424 : OUTPUTS:
13B8 3425 : FORK may be scheduled
13B8 3426 :
13B8 3427 :--
13B8 3428 SCHED_FORK_IO:
OD 68 A5 07 E2 13B8 3429 BBSS #NO_DS_V_FORK_PEND,UCB$$_DEVSTS(R5),10$ ; If BS then fork pending
1451'CF 3F 13BD 3430 PUSHAW W^FORK_DONE ; Set address of fork routine
53 02 9A 13C1 3431 MOVZBL #NO_FS_M_IOFORK,R3 ; Set IO fork
00000000'GF 17 13C4 3432 JMP G^EXES$FORK
13CA 3433
13CA 3434 10$: SETBIT #NO_FS_V_IOFORK,UCB$$_FR3(R5) ; Set port fork pending
05 13CF 3435 RSB
13D0 3436
13D0 3437 SCHED_FORK_POWERFAIL:
OD 68 A5 07 E2 13D0 3438 BBSS #NO_DS_V_FORK_PEND,UCB$$_DEVSTS(R5),10$ ; If BS then fork pending
1451'CF 3F 13D5 3439 PUSHAW W^FORK_DONE ; Set address of fork routine
53 04 9A 13D9 3440 MOVZBL #NO_FS_M_POWERFAIL,R3 ; Set Powerfail fork
```


NODRIVER
V04-001

N 2
- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00
NOSCLASS_PORTFORK - Asyncnh DDCMP fork ro 6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 Page 80
(51)

00000000*GF 17 13DC 3441 JMP G^EXESFORK
13E2 3442
13E2 3443 10\$: SETBIT #NO_FS_V_POWERFAIL,UCB\$L_FR3(R5) ; Set port fork pending
05 13E7 3444 RSB
13E8 3445


```

13E8 3447      .SBTTL NOS$SETUP_UCB - Async DDCMP set up UCB
13E8 3448      :++
13E8 3449      : NOS$SETUP_UCB - Async DDCMP set up UCB
13E8 3450      :
13E8 3451      : FUNCTIONAL DESCRIPTION:
13E8 3452      :
13E8 3453      : This does nothing
13E8 3454      :
13E8 3455      : INPUTS:
13E8 3456      :         R5 = UCB address
13E8 3457      :
13E8 3458      : OUTPUTS:
13E8 3459      :         NONE
13E8 3460      :
13E8 3461      :--
05 13E8 3462 NOS$SETUP_UCB:
13E8 3463      RSB
13E9 3464
  
```



```

13E9 3466 .SBTTL NOSPORT_TRANSITION - Async DDCMP port transition routine
13E9 3467 :++
13E9 3468 : NOSPORT_TRANSITION - Async DDCMP port transition routine
13E9 3469 :
13E9 3470 : FUNCTIONAL DESCRIPTION:
13E9 3471 :
13E9 3472 : This does nothing
13E9 3473 :
13E9 3474 : INPUTS:
13E9 3475 : R1 = Transition type (one of the following)
13E9 3476 : MODEMSC_INIT - initialize the modem control
13E9 3477 : MODEMSC_SHUTDOWN - Shut down the line and protocol
13E9 3478 : MODEMSC_DATASET - Data set signal changes
13E9 3479 : R2 = Type specific argument
13E9 3480 : MODEMSC_DATASET - new receive modem mask
13E9 3481 : R5 = UCB address
13E9 3482 :
13E9 3483 : OUTPUTS:
13E9 3484 : NONE
13E9 3485 :
13E9 3486 :--
05 13E9 3487 NOSPORT_TRANSITION:
13E9 3488 RSB
13EA 3489
  
```



```
13EA 3491
13EA 3492
13EA 3493 .SBTTL NOSREADERROR - Async DDCMP read error routine
13EA 3494 :++
13EA 3495 : NOSREADERROR - Async DDCMP read error routine
13EA 3496 :
13EA 3497 : FUNCTIONAL DESCRIPTION:
13EA 3498 :
13EA 3499 : This routine deciphers the error sent by the port driver into an
13EA 3500 : error code to give to the protocol.
13EA 3501 :
13EA 3502 : INPUTS:
13EA 3503 : R3 = Character and flags:
13EA 3504 : Flags:
13EA 3505 : Bit 12 - parity error on the given character
13EA 3506 : Bit 13 - framing error on the given character
13EA 3507 : Bit 14 - data overrun
13EA 3508 : R5 = UCB address
13EA 3509 :
13EA 3510 : OUTPUTS:
13EA 3511 : NONE
13EA 3512 :
13EA 3513 :--
13EA 3514 NOSREADERROR:
13EA 3515 PUSH R0,R2,R4,R6,R7,R8
13EA 3516 MOVL UCB$NO_BUFFER(R5),R4 ; Get NOB address
13EA 3517 BEQL 10$
13EA 3518 MOVZBL NOB$RSTATE(R4),R0 ; Get receiver state
13EA 3519 $DISPATCH R0,TYPE=B,-
13EA 3520 <- ; Function ; Action
13EA 3521 <NOSC_SEAR MSG NO_ERROR>,-
13EA 3522 <NOSC_HEADER HEADER_ERROR>,-
13EA 3523 <NOSC_DATA DATA_ERROR>,-
13EA 3524 <NOSC_NOBUFFER NOBUFFER_ERROR>,-
13EA 3525 >
13EA 3526 BUG_CHECK NOBUFPCKT,FATAL
13EA 3527 10$:
13EA 3528 NO_ERROR:
13EA 3529 POP R0,R2,R4,R6,R7,R8
13EA 3530 RSB
13EA 3531
13EA 3532 ; If the device is rcvng the header and it did not get an overrun error
13EA 3533 ; then set that this is a header crc error.
13EA 3534
13EA 3535 HEADER_ERROR:
13EA 3536
13EA 3537 ; Get a buffer from the queue of free buffers. If there
13EA 3538 ; is no buffer available then the error cna not be reported.
13EA 3539
13EA 3540 BSBW GET RECEIVE
13EA 3541 BLBC R0,NOBUFFER_ERROR
13EA 3542 BBS #^X,R3,OVERRUN_ERROR
13EA 3543 MOVW #DLK$H_HDRCRC,R3W_ERROR(R2)
13EA 3544 BRB COMP_RCV_ERROR
13EA 3545
13EA 3546 ; Set that the device got a receive overrun error
13EA 3547
```

54 01D5 8F BB 13EA 3515
00A8 C5 D0 13EE 3516
16 13 13F3 3517
50 43 A4 9A 13F5 3518
13F9 3519
13F9 3520
13F9 3521
13F9 3522
13F9 3523
13F9 3524
13F9 3525
1407 3526
140B 3527
140B 3528
01D5 8F BA 140B 3529
05 05 140F 3530
1410 3531
1410 3532
1410 3533
1410 3534
1410 3535
1410 3536
1410 3537
1410 3538
1410 3539
FCC8 30 1410 3540
2B 50 E9 1413 3541
06 53 0E E0 1416 3542
10 A2 02 B0 141A 3543
12 11 141E 3544
1420 3545
1420 3546
1420 3547


```

10 A2 20 B0 1420 3548 OVERRUN_ERROR:
OC 11 1420 3549 MOVW #DLK$M_RCV_OVR,RCV_W_ERROR(R2)
1424 3550 BRB COMP_RCV_ERROR
1426 3551
1426 3552 ; If the device was receiving data then set that this is a data crc error.
1426 3553
1426 3554 DATA_ERROR:
52 30 A4 D0 1426 3555 MOVL NOB$L_RCV_INPR(R4),R2 ; Get receive buffer
F2 53 0E E0 142A 3556 BBS #^XE,R3,OVERRUN_ERROR
10 A2 04 B0 142E 3557 MOVW #DLK$M_DATA_CRC,RCV_W_ERROR(R2)
1432 3558 ;BRB COMP_RCV_ERROR
1432 3559
1432 3560 COMP_RCV_ERROR:
2C B4 30 A4 D4 1432 3561 CLRL NOB$L_RCV_INPR(R4) ; Clear rcv inpr
62 0E 1435 3562 INSQUE (R2),NOB$Q_POST+4(R4) ; Put buffer on queue to comp
54 DD 1439 3563 PUSHL R4
FF7A 30 143B 3564 BSBW SCHED_FORK_IO ; Sched a fork to comp buffer
54 8ED0 143E 3565 POPL R4
1441 3566 NOBUFFER_ERROR:
FC89 30 1441 3567 BSBW START_RECEIVE ; Start up next receive
01D5 8F BA 1444 3568 POPR #^M<R0,R2,R4,R6,R7,R8>
05 1448 3569 RSB
1449 3570

```



```

1449 3572
1449 3573 .SBTTL NOSCLASS_DISCONNECT - Async DDCMP class disconnect
1449 3574 :++
1449 3575 : NOSCLASS_DISCONNECT - Async DDCMP class disconnect
1449 3576 :
1449 3577 : FUNCTIONAL DESCRIPTION:
1449 3578 :
1449 3579 : This does nothing
1449 3580 :
1449 3581 : INPUTS:
1449 3582 : R5 = UCB address
1449 3583 :
1449 3584 : OUTPUTS:
1449 3585 : NONE
1449 3586 :
1449 3587 :--
05 1449 3588 NOSCLASS_DISCONNECT:
1449 3589 RSB
144A 3590
144A 3591
  
```

```

144A 3593 .SBTTL NOSCLASS_POWERACTION - Async DDCMP power action
144A 3594 :++
144A 3595 : NOSCLASS_POWERACTION - Async DDCMP power action
144A 3596 :
144A 3597 : FUNCTIONAL DESCRIPTION:
144A 3598 :
144A 3599 : This routine forks to shutdown the device on a power fail.
144A 3600 :
144A 3601 : INPUTS:
144A 3602 : R5 = UCB address
144A 3603 :
144A 3604 : OUTPUTS:
144A 3605 : NONE
144A 3606 :
144A 3607 :--
FF83 30 144A 3608 NOSCLASS_POWERACTION:
05 144A 3609 BSBW SCHED_FORK_POWERFAIL
144D 3610 RSB
  
```


NODRIVER
V04-001

- VAX/VMS DMF32 Async DDMCP Line Driver H 3 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 87
NOSNULL - Async DDCMP null routine 6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (59)

```
144E 3612 .SBTTL NOSNULL - Async DDCMP null routine
144E 3613 :++
144E 3614 : NOSNULL - Async DDCMP null routine
144E 3615 :
144E 3616 : FUNCTIONAL DESCRIPTION:
144E 3617 :
144E 3618 : This does nothing
144E 3619 :
144E 3620 : INPUTS:
144E 3621 : R5 = UCB address
144E 3622 :
144E 3623 : OUTPUTS:
144E 3624 : NONE
144E 3625 :
144E 3626 :--
05 144E 3627 NOSNULL:
144E 3628 RSB
144F 3629
```

```
144F 3631
144F 3632 .SBTTL FORKDONE - Fork process
144F 3633 :++
144F 3634 : FORKDONE - Fork process
144F 3635 :
144F 3636 : FUNCTIONAL DESCRIPTION:
144F 3637 :
144F 3638 : This routine is entered for two reasons either to complete some IO, or
144F 3639 : because the port has some work that must be done at FORK level.
144F 3640 :
144F 3641 : INPUTS:
144F 3642 :
144F 3643 : R4 = NOB address
144F 3644 : R5 = UCB address
144F 3645 :
144F 3646 : IMPLICIT INPUTS:
144F 3647 : UCB$$_FR3 = Type of fork
144F 3648 :
144F 3649 : IPL = FIPL
144F 3650 :
144F 3651 : OUTPUTS:
144F 3652 : R5 = UCB address
144F 3653 : R0,R6 - R9 are preserved.
144F 3654 :
144F 3655 : If a receive I/O request is pending, the receive is completed.
144F 3656 : Otherwise, queue the message for a future I/O.
144F 3657 :
144F 3658 :--
02C2' 144F 3659 .WORD TIMEOUT-.
1451 3660 FORK_DONE:
1451 3661 DSBINT UCB$$_DIPL(R5) ; Disable device interrupts
1458 3662 CLRBIT #NO_DS_V_FORK_PEND,UCB$$_DEVSTS(R5) ; Clear fork pending
OD 10 A5 00 E5 145D 3663 BBCC #NO_FS_V_PORTFORK,UCB$$_FR3(R5),5$ ; If BC don't call portfork
50 0118 C5 D0 1462 3664 ENBINT
34 B0 16 1465 3665 MOVL UCB$$_TT_PORT(R5),R0 ; Get port vector table
03 11 146A 3666 JSB @PORT_FORKRET(R0) ; Call port fork routine
146D 3667 BRB 10$
146F 3668 5$: ENBINT
1472 3669 10$: DSBINT UCB$$_DIPL(R5) ; Disable device interrupts
1479 3670 CLRBIT #NO_FS_V_IOFORK,UCB$$_FR3(R5)
147E 3671
147E 3672 ; Check for a powerfail and shut down the device if one has happened
147E 3673
42 10 A5 02 E4 147E 3674 BBSC #NO_FS_V_POWERFAIL,UCB$$_FR3(R5),45$
1483 3675 ENBINT
54 00A8 C5 D0 1486 3676 MOVL UCB$$_NO_BUFFER(R5),R4 ; Get NOB address
03C0 8F BB 148B 3677 PUSHF #M<R6,R7,R8,R9> ; Save registers
52 28 B4 0F 148F 3678 15$: REMQUE @NOB$$_POST(R4),R2 ; Get next buffer to complete
1C 1D 1493 3679 BVS 30$ ; If VS then no buffer found
13 0A A2 91 1495 3680 CMPB IRP$$_TYPE(R2),S^#DYN$$_BUFIO ; If NEQ then not XMT buffer
08 12 1499 3681 BNEQ 20$
0151 30 149B 3682 BSBW TRANSMIT_DONE ; Complete the transmit
18 50 E9 149E 3683 BLBC R0,40$ ; If BC then error on buffer
EC 11 14A1 3684 BRB 15$ ; Get next entry to post
17 0A A2 91 14A3 3685 20$: CMPB IRP$$_TYPE(R2),S^#DYN$$_NET ; If EQL then RCV buffer
2E 12 14A7 3686 BNEQ 60$
002F 30 14A9 3687 BSBW RECEIVE_DONE ; Complete the Receive
```


0A 50	E9	14AC	3688		BLBC	R0,40\$		; If BC then error on buffer
DE	11	14AF	3689		BRB	15\$		; Get next entry to post
50 01	3C	14B1	3690	30\$:	MOVZWL	S^#SS\$-NORMAL,R0		; Set status
03C0 8F	BA	14B4	3691		POPR	#^M<R6,R7,R8,R9>		; Restore registers
	05	14B8	3692		RSB			; Return
		14B9	3693					
03C0 8F	BA	14B9	3694	40\$:	POPR	#^M<R6,R7,R8,R9>		; Restore registers
16	E1	14BD	3695		BBC	#XMSV_ERR TRIB,-		; Assume trib shutdown
0D 44 A5		14BF	3696			UCB\$L_DEVDEPEND(R5),50\$		
041B	31	14C2	3697		BRW	SHUTDOWN_CIRCUIT		
		14C5	3698					
		14C5	3699	45\$:	ENBINT			
0082 C5	B6	14C8	3700		INCW	UCB\$W_ERRCNT(R5)		
0212	30	14CC	3701		BSBW	POKE_USER		
		14CF	3702	50\$:	SETBIT	#XMSV_ERR_FATAL,UCB\$L_DEVDEPEND(R5)		; Set fatal error
0377	31	14D4	3703		BRW	SHUTDOWN_CIRCUIT		; Br to shut down the device
		14D7	3704					
		14D7	3705	60\$:	BUG_CHECK	NOBUFPCKT,FATAL		; Else fatal, error
		14DB	3706					

```
14DB 3708 .SBTTL RECEIVE_DONE - Complete a receive buffer
14DB 3709
14DB 3710 :++
14DB 3711 :RECEIVE_DONE
14DB 3712 :
14DB 3713 : FUNCTIONAL DESCRIPTION:
14DB 3714 :
14DB 3715 : When the class driver finishes a receive buffer this routine is called to
14DB 3716 : post the receive. First
14DB 3717 : a call is made to the protocol to strip off the header and check the CRC's
14DB 3718 : to ensure the data made it over the wire ok. If an error is detected then
14DB 3719 : the error is recorded in the protocol and the buffer is returned to the
14DB 3720 : free buffer pool, and NOT given to the user. If no error occurs and the
14DB 3721 : buffer is a protocol control message then the buffer is returned to the free
14DB 3722 : buffer pool. If this is a data message and a IRP is free the buffer is
14DB 3723 : completed with the IRP, else it is put on a queue to await I/O completion.
14DB 3724 :
14DB 3725 : INPUTS:
14DB 3726 : R2 = Message buffer address
14DB 3727 : R4 = NOB address
14DB 3728 : R5 = UCB address
14DB 3729 :
14DB 3730 : OUTPUTS:
14DB 3731 : R0 = Status
14DB 3732 : R2,R3,R4, and R5 are preserved
14DB 3733 :--
14DB 3734 RECEIVE_DONE:
14DB 3735 PUSH R2,R3,R4,R5 : Receive done
14DB 3736 MOVZBL #DLK$RCVMSG,R6 : Save registers
14DB 3737 MOVW RCV_W_ERROR(R2),R7 : Set that this is a RCV
14DB 3738 MOVAL RCV_Z_HEADER(R2),R8 : Set error bits
14DB 3739 CLRL R9 : Get buffer address
14DB 3740 MOVL NOB$A_PRO_BUFFER(R4),R5 : Clear reg
14DB 3741 BSBW DDCMP : Set protocol buffer address
14DB 3742 POP R2,R3,R4,R5 : Restore registers
14DB 3743 MOVW R9,RCV_W_MSGSIZ(R2) : Set transfer size
14DB 3744 CMPB #DLK$ACTNOTCOM,R6 : If EQL then protl not active
14DB 3745 BEQL 40$
14DB 3746 BITW #<DLK$M_PRSTERR!- : If NEQ then fatal protocol
14DB 3747 DLK$M_HDRERR>,R7 : error go to shutdown the
14DB 3748 BNEQ 55$ : circuit
14DB 3749 BITW #<DLK$M_STRTRCV!- : If NEQ then trib error
14DB 3750 DLK$M_MNTRCV>,R7 : go to shutdown the circuit
14DB 3751 BNEQ 60$
14DB 3752 BBS #DLK$V_CRC,R7,40$ : If CRC failed don't complete buff
14DB 3753 BBC #DLK$V_TRNLK,R7,20$ : If BC don't turn link
14DB 3754 BICW #NO_DS_M_XMTING,- : Start transmitter
14DB 3755 UCB$W_DEVSTS(R5)
14DB 3756 PUSH R2,R7 : Save the register
14DB 3757 CLRL R1 : Set no status for start xmt
14DB 3758 BSBW START_TRANSMIT : Start the tranmsit
14DB 3759 POP R2,R7 : Restore the register
14DB 3760 BLBC R0,70$ : Branch on error
14DB 3761 BBC #DLK$V_XMTCMP,R7,23$ : If BC no transmit to complete
14DB 3762 PUSH R2 : Save receive buffer
14DB 3763 BSBW FINISH_XMT_IO : Else branch to post the I/O
14DB 3764 POPL R2 : Restore receive buffer
```

57 56 3C BB 14DB 3735
57 10 A2 B0 14DD 3736
58 40 A2 DE 14E0 3737
58 59 D4 14E4 3738
55 0C A4 D0 14E8 3739
EBOF 30 14EA 3740
3C BA 14EE 3741
OC A2 59 B0 14F1 3742
56 09 91 14F3 3743
4E 13 14F7 3744
57 0A B3 14FA 3745
51 12 14FC 3746
00C0 8F B3 14FF 3747
51 12 14FF 3748
3E 57 10 E0 1501 3749
14 57 0E E1 1506 3750
10 AA 1506 3751
68 A5 1508 3752
0084 8F BB 150C 3753
51 D4 1510 3754
F72A 30 1512 3755
0084 8F BA 1514 3756
50 50 E9 1518 3757
08 57 0A E1 151A 3758
52 DD 151D 3759
00E2 30 1521 3760
52 8ED0 1524 3761
1528 3762
152A 3763
152D 3764

20\$:


```
16 57 08 E1 1530 3765 23$: BBC #DLK$V_RCVACK,R7,40$ ; If BC then not a data message
1534 3766
53 18 B4 0F 1534 3767 REMQUE @NOBSQ_RCVS(R4),R3 ; Remov waiting rcv I/O request
02 1D 1538 3768 BVS 25$ ; VS then no packet to complete
3D 11 153A 3769 BRB FINISH_RCV_IO ; If found then finish the I/O
153C 3770
14 B4 62 0E 153C 3771 25$: INSQUE (R2),@NOBSQ_ATTN+4(R4) ; Else, queue message buffer
54 DD 1540 3772 30$: PUSHL R4
019C 30 1542 3773 BSBW POKE_USER ; Poke the user
54 8ED0 1545 3774 POPL R4
04 11 1548 3775 BRB 45$
154A 3776
24 B4 62 0E 154A 3777 40$: INSQUE (R2),@NOBSQ_FREE+4(R4) ; Return the buff to free list
50 01 3C 154E 3778 45$: MOVZWL S^#SS$_NORMAL,R0 ; Set normal return
05 1551 3779 RSB
1552 3780
1552 3781 55$: SETBIT #XMSV_ERR_FATAL,UCBSL_DEVDEPEND(R5) ; Set that a fatal error occurd
1557 3782
1557 3783
1557 3784 .IF DF ERR$$$
1557 3785 incw ucb$w_xg_prst(r5) ; inc persitant error count
1557 3786 .ENDC ;DF ERR$$$
1557 3787
16 11 1557 3788 58$: BRB 65$
C8 1559 3789 60$: BISL #<XMSM_ERR_START!- ; Assume the trib error occurrd
155A 3790 XMSM_ERR_TRIB>,- ; because a STRT was received
155A 3791 UCB$[ DEVDEPEND(R5)
1561 3792 BBC #DLK$V_MNTRCV,R7,65$ ; If BC then true
1565 3793 CLRBIT #XMSV_ERR_START,UCBSL_DEVDEPEND(R5)
156A 3794 SETBIT #XMSV_ERR_MAINT,UCBSL_DEVDEPEND(R5) ; Else set maint msg rcv'd
50 20D4 8F 3C 156F 3795 65$: MOVZWL #SS$_DEVINACT,R0 ; Set protocol inactive
24 B4 62 0E 1574 3796 70$: INSQUE (R2),@NOBSQ_FREE+4(R4) ; Return the buff to free list
05 1578 3797 RSB
1579 3798
```

```
1579 3800 .SBTTL FINISH_RCV_IO - Finish receive I/O processing
1579 3801 :++
1579 3802 : FINISH_RCV_IO - Finish receive I/O processing
1579 3803 :
1579 3804 : FUNCTIONAL DESCRIPTION:
1579 3805 :
1579 3806 : This routine completes a receive operation that has been matched with a
1579 3807 : message block. After the receive has been completed the message free list is
1579 3808 : filled and a receive is started if needed.
1579 3809 :
1579 3810 : INPUTS:
1579 3811 : R2 = message buffer address
1579 3812 : R3 = I/O packet address
1579 3813 : R4 = NOB address
1579 3814 : R5 = UCB address
1579 3815 :
1579 3816 : IPL = FIPL
1579 3817 :
1579 3818 : OUTPUTS:
1579 3819 : R5 = UCB address
1579 3820 :
1579 3821 : The request is completed via I/O post.
1579 3822 :--
1579 3823 : FINISH_RCV_IO:
1579 3824 : MOVAL RCV_T_DATA(R2),(R2) ; Finish receive I/O request
1579 3825 : MOVZWL RCV_W_MSGSIZ(R2),R0 ; Insert address of the data
1579 3826 : CLRL R1 ; Get size of transfer
1579 3827 : MOVL R2,IRP$S_SVAPTE(R3) ; Assume error
1579 3828 : MOVL IRP$S_MEDIA(R3),4(R2) ; Save block address
1579 3829 : ADDW UCB$W_DEVBUSIZ(R5),- ; Insert saved user VA
1579 3830 : NOB$W_QUOTA(R4) ; Adjust unit quota
1579 3831 : CMPW R0,IRP$W_BCNT(R3) ; Request larger than actual?
1579 3832 : BLEQU 20$ ; Br if no
1579 3833 : MOVZWL IRP$W_BCNT(R3),R0 ; Set size to min. of two sizes
1579 3834 20$: MOVW R0,IRP$W_BCNT(R3) ; Set size to transfer
1579 3835 : ASHL #16,R0,R0 ; Set up status
1579 3836 : BNEQ 25$ ; Br if success
1579 3837 : MOVW #SS$S_CTRLERR,R0 ; Set data path error
1579 3838 : BRB 30$
1579 3839 25$: MOVW S^#SS$S_NORMAL,R0 ; Set success
1579 3840 30$: MOVL R0,IRP$S_MEDIA(R3) ; Set status and size
1579 3841 :
1579 3842 : MOVB #DYN$C_BUFIO,IRP$B_TYPE(R2) ; PSI expects this in all buffs
1579 3843 : BSBB IO_DONE ; Post the I/O request
1579 3844 : BSBW FICLFREELIST ; Return rcv buffer to free q
1579 3845 : RSB ; and start next RCV
1579 3846 :
1579 3847 :
1579 3848 : Complete a transfer I/O operation
1579 3849 :
1579 3850 : TRANSMIT_IO_DONE:
1579 3851 : BBC - ; If BC then not an "Internal"
1579 3852 : XMTQ$B_FLAG(R1),IO_DONE ; IRP, else must dealloc the
1579 3853 : ; buffer used to transmit
1579 3854 : MOVL R1,R0 ; Set to dealloc the buffer
1579 3855 : JSB G^COM$DRVDEALMEM ; Deall the buffer
1579 3856 : IO_DONE: ; Comp a transfer I/O operation
```

62 48 A2 DE 1579 3824
50 0C A2 3C 1579 3825
2C A3 51 D4 1581 3826
04 A2 38 A3 D0 1583 3827
42 A5 A0 1587 3828
40 A4 158C 3829
32 A3 50 B1 158F 3830
50 32 A3 1B 1591 3831
32 A3 50 3C 1595 3832
50 50 10 78 1597 3833
50 0054 8F B0 1598 3834
50 03 11 159B 3835
38 A3 50 B0 159F 3836
0A A2 13 90 15A3 3837
12 10 15A5 3838
FAB4 30 15AA 3839
05 15AC 3840
1583 3841
1587 3842
1589 3843
158C 3844
158D 3845
158D 3846
158D 3847
158D 3848
158D 3849
09 1F A1 E1 158D 3850
50 51 D0 158D 3851
00000000 GF 16 158F 3852
15C2 3853
15C2 3854
15C5 3855
15CB 3856

NODRIVER
V04-001

N 3
- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19
FINISH_RCV_IO - Finish receive I/O proce 6-SEP-1984 16:22:41

VAX/VMS Macro V04-00 Page 93
[DRIVER.SRC]NODRIVER.MAR;2 (63)

50	44	A5	D0	15CB	3857	MOVL	UCB\$DEVDEPEND(R5),-	; Set other info
	3C	A3		15CE	3858		IRP\$-MEDIA+4(R3)	
		07	E1	15D0	3859	BBC	#IRP\$DIAGBUF,-	; Br if no diagnostic buffer
	14	2A	A3	15D2	3860		IRP\$STS(R3),20\$	
80	4C	B3	08	C1	15D5	ADDL3	#8,IRP\$DIAGBUF(R3),R0	; Addr buffer past start time
	00000000	'GF	7D	15DA	3862	MOVQ	G^EXE\$GQ SYTIME,(R0)+	; Insert stop time
80	0082	C5	3C	15E1	3863	MOVZWL	UCB\$W_ERRCNT(R5),(R0)+	; Insert error counter
	00F2		30	15E6	3864	BSBW	NO\$REGDUMP	
	00000000	'GF	17	15E9	3865	JMP	G^COM\$POST	; Post the I/O
				15EF	3866			


```

15EF 3868 .SBTTL TRANSMIT_DONE - Transmit completion routine
15EF 3869
15EF 3870 :++
15EF 3871 : TRANSMIT_DONE - Transmit completion routine
15EF 3872 :
15EF 3873 : FUNCTIONAL DESCRIPTION:
15EF 3874 :
15EF 3875 : This routine is called when a transmit buffer needs completion. If the
15EF 3876 : transmit buffer completed was a protocol control buffer then nothing
15EF 3877 : happens with the buffer. In fact it should be part of an the NOB
15EF 3878 : structure . If the transmit was a data buffer, the routine
15EF 3879 : calls the protocol to deal with the buffer. If the protocol notifies the
15EF 3880 : driver that it has XMT's to send to I/O completion, the driver pulls these
15EF 3881 : buffers off the complete queue and sends them off to complete via COM$POST.
15EF 3882 :
15EF 3883 : INPUTS:
15EF 3884 : R2 = Address of buffer to complete
15EF 3885 : R3 = If error then contains the error from the device
15EF 3886 : R4 = NOB address
15EF 3887 : R5 = UCB address
15EF 3888 :
15EF 3889 : IPL = FIPL
15EF 3890 :
15EF 3891 : OUTPUTS:
15EF 3892 : R4,R5 are preserved
15EF 3893 :
15EF 3894 : --
15EF 3895 TRANSMIT_DONE:
58 52 DO 15EF 3896 MOVL R2,R8 ; Put buffer addr into R8
55 0C A4 DO 15F2 3897 PUSHR #^M<R4,R5> ; Save registers
56 02 9A 15F4 3898 MOVL NOB$A_PRO_BUFFER(R4),R5 ; Set protocol buffer address
57 01 9A 15F8 3899 MOVZBL #DLK$C_XMTMSG,R6 ; Set up to put on RTOQ if the
59 01 9A 15FB 3900 MOVZBL #DLK$M_MSGSENT,R7 ; msg needs to be timed out
E9FD' 59 D4 15FE 3901 CLRL R9
30 30 BA 1600 3902 BSBW DDCMP ; Branch to protocol
02 57 0A E1 1603 3903 POPR #^M<R4,R5> ; Restore registers
50 04 10 1605 3904 BBC #DLK$V_XMTCMP,R7,30$ ; If BC then no XMT's to compl
50 01 3C 1609 3905 BSBB FINISH_XMT_IO ; Else complete the XMT
30$ 30$ 3906 MOVZWL S^#SS$ _NORMAL,R0 ; Set normal return
160E 3907 RSB
160F 3908
160F 3909 FINISH_XMT_IO:
56 03C0 8F BB 160F 3910 PUSHR #^M<R6,R7,R8,R9> ; Save registers
58 0C A4 DO 1613 3911 MOVL NOB$A_PRO_BUFFER(R4),R6 ; Get protocol buffer address
58 28 B6 OF 1617 3912 5$: REMQUE @TFSQ_CMPQ(R6),R8 ; Get the next entry on queue
27 1D 161B 3913 BVS 20$ ; If VS then branch to finish
53 0C A8 DO 161D 3914 MOVL XMTQ$ _IRP(R8),R3 ; Get IRP associated with XMT
50 1A A8 3C 1621 3915 MOVZWL XMTQ$W_MSGSIZE(R8),R0 ; Get transfer size
50 50 06 A2 1625 3916 SUBW2 #NO$C_HEADER_LEN,R0 ; Subtract out the protocol hdr
50 50 10 78 1628 3917 ASHL #16,R0,R0 ; Set up status
50 0054 8F B0 162E 3918 BNEQ 10$ ; Br if data transmitted
50 03 11 1633 3919 MOVW #SS$ _CTRLERR,R0 ; Set device error
50 01 B0 1635 3920 BRB 15$
38 A3 50 DO 1638 3921 10$: MOVW #SS$ _NORMAL,R0 ; Assume success
51 58 DO 163C 3922 15$: MOVL R0,IRP$ _MEDIA(R3)
FF7B 30 163F 3923 MOVL R8,R1 ; Set XMTQ buff address
3924 BSBW TRANSMIT_IO_DONE ; Complete the IO

```


NODRIVER
V04-001

- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 95
TRANSMIT_DONE - Transmit completion rout 6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (65)

50	D3	11	1642	3925	BRB	5\$
03C0	01	3C	1644	3926	MOVZWL	S^#SS\$ NORMAL,R0
	8F	BA	1647	3927	POPR	#^M<R6,R7,R8,R9>
		05	164B	3928	RSB	

; Get next to complete
; Restore registers

```
164C 3930
164C 3931 :++
164C 3932 : DEVTIMER - Device wakeup routine
164C 3933 :
164C 3934 : Functional Description:
164C 3935 :
164C 3936 : This routine is called when the protocol timer every second when the
164C 3937 : protocol timer ticks to start any transmits the protocol may
164C 3938 : have queued as a result of the timer going off.
164C 3939 :
164C 3940 : INPUTS:
164C 3941 : R1 = Error status from protocol specific timer routine
164C 3942 : R5 = Address of the UCB
164C 3943 :
164C 3944 : OUTPUTS:
164C 3945 : R0 = Status of line is LBS then line is up
164C 3946 : LBC then line is down
164C 3947 :
164C 3948 : On error the line is shut down, however the UCB is not
164C 3949 : deallocated
164C 3950 :
164C 3951 :--
164C 3952 :.ENABL LSB
164C 3953 DEVTIMER:
164C 3954 MOVL UCB$$_NO_BUFFER(R5),R4 ; Get NOB address
164C 3955 BBS #XMSV_ERR_FATAL,UCB$$_DEVDEPEND(R5),30$ ; If BS then fatal err
164C 3956 BBS #DLK$V_PRSTERR,R1,30$ ; If BS then prst err shut down
164C 3957 ; the device
164C 3958
164C 3959 : Before we do anything else we should check to see if the xmts and/or
164C 3960 : receives in progress timed out.
164C 3961
164C 3962 DSBINT UCB$$_DIPL(R5)
164C 3963 MOVL G^EXE$GL_ABSTIM,R2
164C 3964 BBC #NO_DS_V_XMT_TIME,UCB$$_DEVSTS(R5),5$ ; If clear time not valid
164C 3965 CMPL R2,NOB$$_XMT_DUETIME(R4) ; If LEQ time not up
164C 3966 BLEQ 5$
164C 3967 BSBW TIMEOUT ; Else timeout message
164C 3968 MOVZBL UCB$$_TT_SPEED(R5),R0 ; Reset due time
164C 3969 MOVZWL DUETIME_TABLE[R0],R0
164C 3970 ADDW3 R0,R2,NOB$$_XMT_DUETIME(R4)
164C 3971 5$: BBC #NO_DS_V_RCV_TIME,UCB$$_DEVSTS(R5),8$ ; If clear time not valid
164C 3972 CMPL R0,NOB$$_RCV_DUETIME(R4) ; If LEQ time not up
164C 3973 BLEQ 8$
164C 3974 BSBW TIMEOUT ; Else timeout message
164C 3975 MOVZBL UCB$$_TT_SPEED+1(R5),R0 ; Reset due time
164C 3976 BNEQ 6$
164C 3977 MOVZBL UCB$$_TT_SPEED(R5),R0
164C 3978 6$: MOVZWL DUETIME_TABLE[R0],R0
164C 3979 ADDW3 R0,R2,NOB$$_RCV_DUETIME(R4)
164C 3980 8$: ENBINT
164C 3981 MOVZWL S^#SS$$_NORMAL,R0 ; Assume everything ok
164C 3982
164C 3983 : If the protocol timer has expired then clear the XMTING flag so that
164C 3984 : the control message which must be sent can be. This is true when the
164C 3985 : device is running half duplex. If the message with the select flag is dropped
164C 3986 : then we must be sure that we can send another message with a select flag.
```

54 00AB C5 D0 164C 3954
7A 44 A5 10 E0 1651 3955
76 51 01 E0 1656 3956
165A 3957
165A 3958
165A 3959
165A 3960
165A 3961
52 00000000'GF D0 1661 3962
19 68 A5 0B E1 1668 3963
3C A4 52 D1 166D 3964
13 15 1671 3965
009B 30 1673 3966
50 00F4 C5 9A 1676 3967
50 EA84 CF40 3C 167B 3968
3C A4 52 50 A1 1681 3969
20 68 A5 0C E1 1686 3970
38 A4 50 D1 168B 3971
1A 15 168F 3972
007D 30 1691 3973
50 00F5 C5 9A 1694 3974
05 12 1699 3975
50 00F4 C5 9A 169B 3976
50 EA5F CF40 3C 16A0 3977
38 A4 52 50 A1 16A6 3978
50 01 3C 16AE 3979
16AB 3980
16B1 3981
16B1 3982
16B1 3983
16B1 3984
16B1 3985
16B1 3986


```

05 51 0F E1 16B1 3987
                  16B1 3988 BBC #DLK$V_TMREXPD,R1,10$
                  16B5 3989 CLRBIT #NO_DS_V_XMTING,UCB$W_DEVSTS(R5)
                    51 DD 16BA 3990 10$: PUSH R1
                  F588 30 16BC 3991 BSBW START_TRANSMIT ; Else try to start a transmit
                    51 8ED0 16BF 3992 POPL R1
                  OB 50 E9 16C2 3993 BLBC R0,30$ ; If LBC then fatal error
51 8000 8F AA 16C5 3994 BICW #DLK$M_TMREXPD,R1 ; Clear timer expired
                    05 16CA 3995 20$: RSB
                  16CB 3996
                  16CB 3997 ;!!!!!!!!!!!!!!!!!!!! WHAT TO DO !!!!!!!!!!!!!!!!!!!!!
                  16CB 3998
                  16CB 3999 25$: SETBIT #XMSV_STS_DISC,UCB$L_DEVDEPEND(R5) ; Set modem disconnect sts
                  16D0 4000
                  16D0 4001 ; TFB/GFB blocks are deallocated on shutdown. Make sure R0 is
                  16D0 4002 ; clear so that the timer routine does not try to access them.
                  16D0 4003
                  16D0 4004 30$: SETBIT #XMSV_ERR_FATAL,UCB$L_DEVDEPEND(R5) ; Set fatal error
0176 30 16D5 4005 BSBW SHUTDOWN_LINE ; Shut down the device
                    50 D4 16D8 4006 CLRL R0
                    05 16DA 4007 RSB
                  16DB 4008 .DSABL LSB
                  16DB 4009

```

```

16DB 4011 .SBTTL NOSREGDUMP - Error log and diagnostics register dump
16DB 4012 :++
16DB 4013 : NOSREGDUMP -Diagnostics register dump routine
16DB 4014 :
16DB 4015 : Functional description:
16DB 4016 :
16DB 4017 : This routine is used to return ?????
16DB 4018 :
16DB 4019 : INPUTS:
16DB 4020 : R0 = ADDRESS OF THE BUFFER
16DB 4021 : R5 = UCB ADDRESS OF THE UNIT
16DB 4022 :
16DB 4023 : OUTPUTS:
16DB 4024 : R0,R1 ARE USED
16DB 4025 : R5 = UCB ADDRESS OF THE UNIT
16DB 4026 :--
16DB 4027 NOSREGDUMP:
00 44 0B E1 16DB 4028 BBC #XMSV_STS_ACTIVE,- ; BR if not active
A5 16DD 4029 UCB$L_DEVDEPEND(R5),20$
05 16E0 4030 20$: RSB
16E1 4031

```



```

16E1 4033      .SBTTL  Poke user process on attention condition
16E1 4034
16E1 4035      :++
16E1 4036      : POKE_USER - Poke user process on attention condition
16E1 4037
16E1 4038      : FUNCTIONAL DESCRIPTION:
16E1 4039
16E1 4040      : This routine is used when data is available or the unit is shutdown.
16E1 4041      : The action is to declare the ast's and send a message to the assoc. mailbox.
16E1 4042
16E1 4043      : INPUTS:
16E1 4044      : R4 = Message type -- 0 if no message
16E1 4045      : R5 = Unit UCB address
16E1 4046
16E1 4047      : OUTPUTS:
16E1 4048      : R0 = Low bit clear only if user is not notified
16E1 4049      : R5 = UCB ADDRESS
16E1 4050      :--
16E1 4051      POKE_USER:
16E1 4052      DSBINT  UCB$B_FIPL(R5)          ; POKE USER
16E1 4053      CLRL   -(SP)                    ; Sync to Fork
16E1 4054      MOVAB  UCB$L_NO_AST(R5),R1      ; Assume failure
16E1 4055      TSTL   (R1)                      ; Get AST listhead
16E1 4056      BEQL   17$                      ; Empty ?
16E1 4057      INCL   (SP)                    ; If so, branch
16E1 4058      MOVL   R1,R4                    ; Indicate success
16E1 4059      10$:  MOVL   (R1),R1           ; Copy list head address
16E1 4060      BEQL   15$                      ; Address a block
16E1 4061      MOVL   UCB$L_DEVDEPEND(R5),^X01C(R1) ; If EQL then done
16E1 4062      BRB    10$                      ; Change param
16E1 4063      15$:  JSB    G^COM$DELATTNAST    ; Deliver AST'S
16E1 4064      17$:  POPL   R0                ; Set status
16E1 4065      ENBINT
16E1 4066      RSB

```

51 00AC 7E D4 16E8 4053
61 D5 16EA 4054
17 13 16EF 4055
6E D6 16F1 4056
54 51 D0 16F3 4057
51 61 D0 16F5 4058
07 13 16F8 4059 10\$:
1C A1 44 A5 D0 16FB 4060
F4 11 16FD 4061
00000000 GF 16 1702 4062
50 8ED0 1704 4063 15\$:
05 170A 4064 17\$:
170D 4065
1710 4066

```

1711 4068 .SBTTL TIMEOUT - TIMEOUT
1711 4069 :++
1711 4070 : TIMEOUT - Timeout
1711 4071 :
1711 4072 : FUNCTIONAL DESCRIPTION:
1711 4073 :
1711 4074 : This routine is entered on device timeout. The action is to shut
1711 4075 : the unit down.
1711 4076 :
1711 4077 : INPUTS:
1711 4078 : R5 = UCB address
1711 4079 :
1711 4080 : OUTPUTS:
1711 4081 : NONE
1711 4082 :
1711 4083 : IPL = Device IPL
1711 4084 :--
1711 4085 : TIMEOUT:
1711 4086 : BBC #XMSV_STS_ACTIVE,- ; TIMEOUT OR POWERFAIL
1711 4087 : UCB$C_DEVDEPEND(R5),20$ ; Br BC device is not active
1711 4088 10$: BBC #UCBSV_POWER,- ; Br unless powerfail
1711 4089 : UCB$W_STS(R5),30$
1711 4090 : BSBW SCHED_FORK_POWERFAIL ; Create the error fork process
1711 4091 20$: RSB
1711 4092 :
1711 4093 : Because some devices are flaky when driven as hard as DECnet drives the line,
1711 4094 : we have added code here to clear the device in case it gets stuck. A side
1711 4095 : effect of clearing the device is that any transmit data it had outstanding
1711 4096 : gets lost. In order to handle that condition we will transmit enough pads
1711 4097 : to make up for the size of the message that we feel has been dropped. This
1711 4098 : is decided by the state the device is in and the type of message is was
1711 4099 : transmitting.
1711 4100 :
1711 4101 : There is another case we must worry about here and that is if the reply
1711 4102 : timeout timer has gone off. In that case the protocol takes the xmt away
1711 4103 : from the device, so that messages are not retransmitted out of order.
1711 4104 : Thus we will check for this condition (the NOB$XMT_INPR field is clear)
1711 4105 : and if it occurs then we will send out pads to fill the message length,
1711 4106 : with our best guess. We also put the ctrl msg address in the inpr field
1711 4107 : to fake the xmt into thinking that the message is still active.
1711 4108 :
1711 4109 30$: INCW UCB$W_ERRCNT(R5) ; Incr the error counter
1711 4110 : PUSH R2,R3,R4,R5
1711 4111 : MOVL UCB$L_IT_PORT(R5),R2 ; Get the port vector
1711 4112 : JSB @PORT_ABORT(R2) ; Reset the device
1711 4113 : JSB @PORT_RESUME(R2)
1711 4114 : MOVL UCB$L_NO_BUFFER(R5),R4 ; Get the no buffer
1711 4115 : CLRL R3
1711 4116 : MOVZBL NOB$XSTATE(R4),R1 ; Get old state
1711 4117 : MOVB #NO$C_XMTERR,NOB$XSTATE(R4) ; Set new state
1711 4118 : $DISPATCH R1,TYPE=B,-
1711 4119 : <- ;function ;action
1711 4120 : <NO$C_EMPTY 40$>,-
1711 4121 : <NO$C_HEADER TIMEOUT_HEADER>,-
1711 4122 : <NO$C_HCRC TIMEOUT_HCRC>,-
1711 4123 : <NO$C_DATA TIMEOUT_DATA>,-
1711 4124 : <NO$C_DCRC TIMEOUT_DCRC>,-

```



```
173F 4125 <NOSC_PADS TIMEOUT_PADS>,-
173F 4126 <NOSC_XMTERR TIMEOUT_XMTERR>,-
173F 4127 >
1759 4128
42 A4 94 1759 4129 40$: CLRB NO$B_XSTATE(R4) ; Reset state
3E BA 175C 4130 POPR #^M<RT,R2,R3,R4,R5>
05 175E 4131 RSB
175F 4132
175F 4133 TIMEOUT_HEADER:
53 42 A5 0A A1 175F 4134 ADDW3 #10,UCB$W_DEVBUFSIZ(R5),R3 ; Assume xmt is gone
51 34 A4 D0 1764 4135 MOVL NO$B_XMT_INPR(R4),R1
57 13 1768 4136 BEQL TIMEOUT_END
07 1F A1 02 E0 176A 4137 BBS #XMTQ$V_CONTROL,XMTQ$B_FLAG(R1),10$
176F 4138
176F 4139 ; Set send field high enough to cover xmtng of header, hcrc, data, and dcrc.
53 1A A1 04 A1 176F 4140
4B 11 1774 4142 ADDW3 #4,XMTQ$W_MSGSIZE(R1),R3
1776 4143 BRB TIMEOUT_END
1776 4144 ; Set send length field high enough to cover xmtng of header and hcrc
53 06 B0 1776 4145
46 11 1779 4147 10$: MOVW #NOSC_HEADER_LEN,R3
177B 4148 BRB TIMEOUT_END
177B 4149 TIMEOUT_HCRC:
53 42 A5 04 A1 177B 4150 ADDW3 #4,UCB$W_DEVBUFSIZ(R5),R3 ; Assume xmt is gone
51 34 A4 D0 1780 4151 MOVL NO$B_XMT_INPR(R4),R1
3B 13 1784 4152 BEQL TIMEOUT_END
0A 1F A1 02 E0 1786 4153 BBS #XMTQ$V_CONTROL,XMTQ$B_FLAG(R1),10$
178B 4154
178B 4155 ; Set send field high enough to cover xmtng of hcrc, data, and dcrc.
53 1A A1 04 A1 178B 4156
53 06 A2 1790 4158 ADDW3 #4,XMTQ$W_MSGSIZE(R1),R3
2C 11 1793 4159 SUBW2 #NOSC_HEADER_LEN,R3 ; XMTQ$W_MSGSIZ includes HEADER size
1795 4160 BRB TIMEOUT_END
1795 4161 ; Set send length field high enough to cover xmtng of hcrc
53 02 B0 1795 4162
27 11 1798 4164 10$: MOVW #2,R3
179A 4165 BRB TIMEOUT_END
179A 4166 TIMEOUT_DATA:
53 42 A5 02 A1 179A 4167 ADDW3 #2,UCB$W_DEVBUFSIZ(R5),R3 ; Assume xmt is gone
51 34 A4 D0 179F 4168 MOVL NO$B_XMT_INPR(R4),R1
1C 13 17A3 4169 BEQL TIMEOUT_END
17A5 4170
17A5 4171 ; Set send field high enough to cover xmtng of data, and dcrc.
53 1A A1 02 A1 17A5 4172
53 06 A2 17AA 4174 ADDW3 #2,XMTQ$W_MSGSIZE(R1),R3
12 11 17AD 4175 SUBW2 #NOSC_HEADER_LEN,R3 ; XMTQ$W_MSGSIZ includes HEADER size
17AF 4176 BRB TIMEOUT_END
53 02 B0 17AF 4177 TIMEOUT_DCRC:
0D 11 17AF 4178 MOVW #2,R3 ; Set the field to send DCRC
17B2 4179 BRB TIMEOUT_END
17B4 4180
17B4 4181 TIMEOUT_PADS:
```

```

42 53 02 B0 17B4 4182      MOVW    #2,R3      ; Set the field to send DCRC
    A4 05 90 17B7 4183      MOVB    #NOSC_PADS,NOBSB_XSTATE(R4) ; Set to finish message
    04 11 17BB 4184      BRB      TIMEOUT_END
    17BD 4185
53 46 A4 B0 17BD 4186 TIMEOUT_XMTERR:
    17BD 4187      MOVW    NOBSW_XMTERR_SIZE(R4),R3
    17C1 4188      ;BRB      TIMEOUT_END
    17C1 4189
    17C1 4190 TIMEOUT_END:
    17C1 4191
    34 A4 D5 17C1 4192      TSTL     NOBSL_XMT_INPR(R4)      ; If NEQ then message available
    06 12 17C4 4193      BNEQ     10$
34 A4 0080 C4 DE 17C6 4194      MOVAL   NOBSZ_CTL_MSG(R4),NOBSL_XMT_INPR(R4) ; Else set to fake xmt
    46 A4 53 B0 17CC 4195 10$:      MOVW    R3,NOBSW_XMTERR_SIZE(R4) ; Set size to xmt
011C C5 44 A4 DE 17D0 4196      MOVAL   NOBSW_PADS(R4),UCBSL_TT_OUTADR(R5) ; set to send pads
    0120 C5 02 9B 17D6 4197      MOVZBW  #2,UCBSW_TT_OUTLEN(R5) ; set length
    010B C5 01 8E 17DB 4198      MNEGB   #1,UCBSB_TT_OUTTYPE(R5) ; Set to send a burst
    3E BA 17E0 4199      POPR     #^M<R1,R2,R3,R4,R5>
    05 17E2 4200      RSB
    17E3 4201
```



```
17E3 4203 .SBTTL NOSCANCEL - Cancel I/O routine
17E3 4204 :++
17E3 4205 : NOSCANCEL - Cancels all I/O in progress
17E3 4206 :
17E3 4207 : If the NO buffer has not been allocated that means the device is
17E3 4208 : not running, set normal return to decrement the reference count
17E3 4209 : and return. This condition can happen when the line gets switched
17E3 4210 : from a static terminal port to a static ddcmp port.
17E3 4211 : By looking at the channel number saved at line startup time, cancel
17E3 4212 : determines which entity (line or circuit) to shut down. Only on the
17E3 4213 : last deassign to the channels does the buffer allocated for the
17E3 4214 : TT UCB's extension get deallocated.
17E3 4215 :
17E3 4216 : INPUTS:
17E3 4217 : R2 = channel number
17E3 4218 : R3 = current IRP address
17E3 4219 : R4 = PCB address
17E3 4220 : R5 = UCB address
17E3 4221 : R8 = Cancel reason code (zero vanilla flavored cancel)
17E3 4222 :
17E3 4223 : IPL = FIPL
17E3 4224 :
17E3 4225 : OUTPUTS:
17E3 4226 : R0 - R3 are destroyed
17E3 4227 :
17E3 4228 :--
17E3 4229 NOSCANCEL:
50 00A8 C5 D0 17E3 4230 MOVL UCBSL_NO_BUFFER(R5),R0 ; Get NOB address
23 13 17E8 4231 BEQL 30$ ; If EQL device not active
60 A4 58 A0 D1 17EA 4232 CMPL NOBSL_PID(R0),PCBSL_PID(R4) ; If not starters PID then
1C 12 17EF 4233 BNEQ 30$ ; do not process cancel
52 5C A0 B1 17F1 4234 CMPW NOBSW_CHANL(R0),R2 ; If EQL match on line channel
05 13 17F5 4235 BEQL 10$ ; Else shutdown the circuit
00E6 30 17F7 4236 BSBW SHUTDOWN_CIRCUIT
03 11 17FA 4237 BRB 15$
004F 30 17FC 4238 10$: BSBW SHUTDOWN_LINE
50 DD 17FF 4239 15$: PUSHL R0
5C A5 B5 1801 4240 TSTW UCBSW_REFC(R5) ; If reference count is nonzero
03 12 1804 4241 BNEQ 20$ ; do not deallocate nob or disc port
0008 30 1806 4242 BSBW CLEAR_NO_BUFFER ; Else do
50 8ED0 1809 4243 20$: POPL R0 ; Set status from shutdown
05 180C 4244 RSB
50 01 3C 180D 4245 30$: MOVZWL #SS$_NORMAL,R0
05 1810 4246 RSB
1810 4247
```

```
1811 4249 .SBTTL CLEAR_NO_BUFFER - Clear NO buffer and disconnect the port
1811 4250 :++
1811 4251 : CLEAR_NO_BUFFER - Clear NO buffer and disconnect the port
1811 4252 :
1811 4253 : This routine is called on the last deassign to the device. It deallocates
1811 4254 : the NO buffer which was used as an extension to the devices UCB. And
1811 4255 : it disconnects with out deallocating the UCB from the port.
1811 4256 :
1811 4257 : INPUTS:
1811 4258 : R4 = PCB address
1811 4259 : R5 = UCB address
1811 4260 :
1811 4261 : IPL = FIPL
1811 4262 :
1811 4263 : OUTPUTS:
1811 4264 : All registers are preserved.
1811 4265 :
1811 4266 :--
1811 4267 CLEAR_NO_BUFFER:
1811 4268 PUSHF #^M<R1,R2,R3,R4,R5,R6,R7>
50 00FE 8F BB 1815 4269 MOVL UCB$NO_BUFFER(R5),R0 ; If EQL buffer no allocated
00A8 C5 D0 181A 4270 BEQL 10$
00A8 C5 D4 181C 4271 CLRL UCB$NO_BUFFER(R5) ; Set buffer no longer available
57 58 A0 D0 1820 4272 MOVL NOB$PID(R0),R7 ; Save starters' PID
52 08 A0 3C 1824 4273 MOVZWL UCB$SIZE(R0),R2 ; get size of buffer
00000000'GF 34 BB 1828 4274 PUSHF #^M<R2,R4,R5>
16 182A 4275 JSB G^COM$DRVDEALMEM ; Deallocate the block
BA 1830 4276 POPR #^M<R2,R4,R5>
1832 4277
1832 4278 ; Call the port driver to disconnect the port. Do not delete the UCB.
1832 4279
51 0118 C5 D0 1832 4280 MOVL UCB$TT_PORT(R5),R1
04 B1 16 1837 4281 JSB @PORT_DISCONNECT(R1)
60 A4 57 D1 183A 4282 CMPL R7,PCB$PID(R4) ; Current PID same as starter's PID
09 12 183E 4283 BNEQ 10$ ; NEQ then don't restore quota
50 0080 C4 D0 1840 4284 MOVL PCB$JIB(R4),R0 ; Get JIB
20 A0 52 C0 1845 4285 ADDL2 R2,JIB$BYTCNT(R0) ; Restore the byte count
00FE 8F BA 1849 4286
05 1849 4287 10$: POPR #^M<R1,R2,R3,R4,R5,R6,R7>
184D 4288 RSB
184E 4289
```



```
184E 4291
184E 4292 .SBTTL SHUTDOWN - Shut down unit, device and/or protocol
184E 4293 :++
184E 4294 : SHUTDOWN_LINE - Shut down unit
184E 4295 :
184E 4296 : FUNCTIONAL DESCRIPTION:
184E 4297 :
184E 4298 : This routine is used to shut down the unit as a result of a
184E 4299 : setmode shutdown on the controller or a fatal error. The mapping
184E 4300 : registers are returned, a call is made to shutdown the circuit; and
184E 4301 : all quotas are returned. If this was the last deassign of a channel
184E 4302 : to the device then the NOB is deallocated.
184E 4303 :
184E 4304 : INPUTS:
184E 4305 : R5 = UCB address
184E 4306 :
184E 4307 : For SHUTDOWN_LINE_ALT
184E 4308 :
184E 4309 : R1 = 0 if DTR can be dropped on shut down
184E 4310 : 1 if DTR can not be dropped on shut down
184E 4311 :
184E 4312 : OUTPUTS:
184E 4313 : R5 = UCB address
184E 4314 : R6 - R9 are preserved
184E 4315 : R0-R3 are destroyed.
184E 4316 :--
184E 4317 SHUTDOWN_LINE: ; Shut down unit
184E 4318 MOVL #1,R1 ; Do not deallocate NOB
1851 4319 SHUTDOWN_LINE_ALT:
1851 4320 BBC #NO_DS_V_INITED,- ; BR if not inited
1853 4321 UCB$W_DEVSTS(R5),5$
1856 4322 BBS #UCB$V_ONLINE,- ; Br if online
1858 4323 UCB$W_STS(R5),10$ ; Set status
185B 4324 5$: MOVZWL #SS$_DEVINACT,R0 ; ... and return
1860 4325 RSB
1861 4326
1861 4327 10$: PUSHR #^M<R2,R3,R4,R6,R7,R8,R9> ; Save the registers
1865 4328
1865 4329 BICW #UCB$M_POWER!- ; Reset device status
1869 4330 UCB$M_TIM,UCB$W_STS(R5) ; Clear all but xmt off
1869 4331 BICW #^C<NO_DS_M_XMTING!- ; rcv off
186A 4332 NO_DS_M_RCVING!- ; internal loop
186A 4333 NO_DS_M_ILOOP_SUP>,-
186A 4334 UCB$W_DEVSTS(R5)
186F 4335
186F 4336 ; Shut down the circuit
186F 4337
186F 4338
186F 4339 PUSHL R1 ; Save NOB deallocate status
1871 4340 BSBW SHUTDOWN_CIRCUIT ; Shutdown the circuit
1874 4341 MOVL UCB$L_NO_BUFFER(R5),R4 ; Get NOB address
1879 4342
1879 4343 ; Stop the DDCMP timer. First set up registers with params for call
1879 4344 ; then make the call to the protocol.
1879 4345
1879 4346 PUSHR #^M<R4,R5>
187B 4347 MOVZBL #DLK$_STOP_TIMER,R6 ; Set up R6 with the DDCMP comm
```

```

57 7C 187E 4348 CLRQ R7 ; CLR all registers
59 D4 1880 4349 CLRL R9 ; Clear R9
55 0C A4 D0 1882 4350 MOVL NOB$A_PRO_BUFFER(R4),R5 ; Set up R5 with TF block addr
50 E777' 30 1886 4351 BSBW DDCMP ; Call protocol
55 D0 1889 4352 MOVL R5,R0 ; Need to deallocate protocol buffer
30 BA 188C 4353 POPR #^M<R4,R5>
188E 4354 ;
188E 4355 ; After the timer has been stopped it is ok to reload the driver.
188E 4356 ;
0000000D'EF 04 8A 188E 4357 BICB #DPT$M_NOUNLOAD,DPT$TAB+DPT$B_FLAGS
1895 4358 ;
1895 4359 ; Deallocate the protocol buffer
1895 4360 ;
52 0C A4 D4 1895 4361 CLRL NOB$A_PRO_BUFFER(R4) ; Clear all knowledge of protocol bu
08 A0 3C 1898 4362 MOVZWL UCB$W_SIZE(R0),R2 ; Get the size
34 BB 189C 4363 PUSHR #^M<R2,R4,R5>
00000000'GF 16 189E 4364 JSB G^COM$DRVDEALMEM ; Dealloc the buffer
34 BA 18A4 4365 POPR #^M<R2,R4,R5>
18A6 4366 ;
18A6 4367 ;
18A6 4368 ; Restore the buffered I/O quota and protocol buffer quota to the starter
18A6 4369 ;
51 50 58 A4 3C 18A6 4370 MOVZWL NOB$L_PID(R4),R0 ; Get pid of last starter
00000000'GF D0 18AA 4371 MOVL G^SCH$GL_PCBVEC,R1 ; Address PCB vector
50 6140 D0 18B1 4372 MOVL (R1)[R0],R0 ; Get PCB of owner
56 60 A0 D0 18B5 4373 MOVL PCB$L_PID(R0),R6 ; Save pid of PCB
57 58 A4 D0 18B9 4374 MOVL NOB$L_PID(R4),R7 ; Save PID of starter
51 40 A4 3C 18BD 4375 MOVZWL NOB$W_QUOTA(R4),R1 ; Get quota
40 A4 B4 18C1 4376 CLRW NOB$W_QUOTA(R4) ; Clear quota returned
52 51 C0 18C4 4377 ADDL R1,R2 ; Get bytes quota to return
51 8ED0 18C7 4378 POPL R1 ; Restore drop DTR status
57 56 D1 18CA 4379 CMPL R6,R7 ; Do the PIDs match
09 12 18CD 4380 BNEQ 20$ ; If NEQ no don't return quota
50 0080 C0 D0 18CF 4381 MOVL PCB$L_JIB(R0),R0 ; Get JIB
20 A0 52 C0 18D4 4382 ADDL R2,JIB$L_BYTCNT(R0) ; Return byte count quota
50 01 3C 18D8 4383 MOVZWL S^#SS$_NORMAL,R0 ; Set status
03DC 8F BA 18DB 4384 POPR #^M<R2,R3,R4,R6,R7,R8,R9> ; Save the registers
05 18DF 4385 RSB
18E0 4386
18E0 4387
```



```
18E0 4389
18E0 4390 :++
18E0 4391 : SHUTDOWN_CIRCUIT - Shut down device and protocol
18E0 4392 :
18E0 4393 : FUNCTIONAL DESCRIPTION:
18E0 4394 :
18E0 4395 : This routine is used to shut down the circuit as a result of a setmode
18E0 4396 : shutdown on the tributary or on the controller or by a fatal error on
18E0 4397 : the device. The routines frees allocated blocks; completes IRP's with
18E0 4398 : the active bit clear; and halts the protocol.
18E0 4399 :
18E0 4400 : INPUTS:
18E0 4401 : R5 = UCB address
18E0 4402 :
18E0 4403 : OUTPUTS:
18E0 4404 : R5 = UCB address
18E0 4405 : R6 - R9 are preserved
18E0 4406 : R0-R3 are destroyed.
18E0 4407 :--
18E0 4408 SHUTDOWN_CIRCUIT:
06 44 A5 0B E4 18E0 4409 BBSC #XMSV_STS_ACTIVE,UCB$$_DEVDEPEND(R5),10$ ; If BS then trib is active
50 20D4 8F 3C 18E5 4410 MOVZWL #SS$_DEVINACT,R0 ; Else set status and return
05 18EA 4411 RSB
18EB 4412
68 A5 03DC 8F BB 18EB 4413 10$: PUSHR #^M<R2,R3,R4,R6,R7,R8,R9> ; Save the registers
A8 18EF 4414 BISW #<NO_DS_M_XMTING!- ; Set the XMTer and
18F5 4415 NO_DS_M_RCVING!- ; RCVer off
18F5 4416 NO_DS_M_SHUTDOWN>,UCB$$_DEVSTS(R5) ; and set device is shutting dow
18F5 4417
18F5 4418 : What we are about to do is in direct violation of the DDCMP standard.
18F5 4419 : But as it is we can not think of what else to do. If anyone out there
18F5 4420 : has a better suggestion please feel free to send it to us. We are however
18F5 4421 : going to abort the receiver and transmitter in the next few instructions.
18F5 4422 : The ddcmp standard says that the transmitter should not be aborted, but
18F5 4423 : in order to make this true under all conditions we would have to
18F5 4424 : wait at this spot for as much as 90 seconds.
18F5 4425
18F5 4426 DSBINT UCB$$_DIPL(R5)
52 0118 C5 D0 18FC 4427 MOVL UCB$$_TT_PORT(R5),R2 ; Get the port vector
20 B2 16 1901 4428 JSB @PORT_ABORT(R2) ; Reset the device
24 B2 16 1904 4429 JSB @PORT_RESUME(R2)
54 00A8 C5 D0 1907 4430 MOVL UCB$$_NO_BUFFER(R5),R4 ; Get NOB address
42 A4 94 190C 4431 CLRB NOB$$_XSTATE(R4) ; Reset transmitter state
F7BB 30 190F 4432 BSBW START_RECEIVE ; Reset receiver state
1912 4433
1912 4434
1912 4435 : Clear any receives or transmits currently inprogress
1912 4436
56 30 A4 D0 1912 4437 MOVL NOB$$_RCV_INPR(R4),R6 ; If EQL then on receive pending
07 13 1916 4438 BEQL 20$
30 A4 D4 1918 4439 CLRL NOB$$_RCV_INPR(R4) ; Clear receive from inprogress
24 B4 66 0E 191B 4440 INSQUE (R6),@NOB$$_FREE+4(R4) ; Queue to free queue
51 34 A4 D0 191F 4441 20$: MOVL NOB$$_XMT_INPR(R4),R1 ; Comp XMT's on inprogress Q
0C 13 1923 4442 BEQL 30$ ; If EQL none to complete
34 A4 D4 1925 4443 CLRL NOB$$_XMT_INPR(R4) ; Clear transmit from inprogress
02 E0 1928 4444 BBS #XMTQ$V_CONTROL,- ; If BS then control message
04 1F A1 192A 4445 XMTQ$B_FLAG(R1),30$ ; don't put on post queue
```



```
2C B4 61 0E 192D 4446      INSQUE (R1),@NOBSQ_POST+4(R4)      ; Insert to be completed
1931 4447 30$: ENBINT      ; Return to fork level
1934 4448
1934 4449 ; Deallocate all the attention AST control blocks
1934 4450
57 00AC 54 DD 1934 4451      PUSHL R4
50 67 9E 1936 4452 40$: MOVAB UCB$$_NO_AST(R5),R7      ; Address list head for AST's
1B 13 DO 193B 4453      MOVL (R7),R0      ; Anything in the list?
56 22 A0 3C 193E 4454      BEQL 45$      ; If EQL then empty
52 24 A0 3C 1940 4455      MOVZWL ACB$$_KAST+10(R0),R6      ; Force channel
54 00000000 GF DO 1944 4456      MOVZWL ACB$$_KAST+12(R0),R2      ; Get PID index
54 54 64 42 DO 1948 4457      MOVL G^SCH$$_GL_PCBVEC,R4
00000000 GF DO 194F 4458      MOVL (R4)[R2],R4      ; Get PCB
DB 11 1953 4459      JSB G^COM$$_FLUSHATTNS      ; Flush the attention AST's
54 8ED0 195B 4460      BRB 40$      ; Continue until done
195E 4461 45$: POPL R4
195E 4462
195E 4463 ; Clear the post queue
195E 4464
56 28 B4 0F 195E 4465 50$: REMQUE @NOBSQ_POST(R4),R6      ; Get next buffer to complete
24 1D 1962 4466      BVS 60$      ; If VS then no buffer found
13 0A A6 91 1964 4467      CMPB IRP$$_TYPE(R6),S^#DYN$$_BUF10      ; If NEQ then not XMT buffer
12 12 1968 4468      BNEQ 55$
53 0C A6 DO 196A 4469      MOVL XMTQ$$_IRP(R6),R3      ; If EQL then no IRP to comp
EE 13 196E 4470      BEQL 50$
51 56 DO 1970 4471      MOVL R6,R1      ; Set up R1 for branch
38 A3 2C 3C 1973 4472      MOVZWL #SS$$_ABORT,IRP$$_MEDIA(R3)      ; Set abort status and compl
FC 43 30 1977 4473      BSBW TRANSMIT_IO_DONE      ; the request
E2 11 197A 4474      BRB 50$
17 0A A6 91 197C 4475 55$: CMPB IRP$$_TYPE(R6),S^#DYN$$_NET      ; If EQL then RCV buffer
2D 12 1980 4476      BNEQ 85$
24 B4 66 0E 1982 4477      INSQUE (R6),@NOBSQ_FREE+4(R4)      ; Put on free Q to complete
D6 11 1986 4478      BRB 50$
1988 4479
1988 4480 ; Deallocate all receive buffers
1988 4481
56 20 A4 9E 1988 4482 60$: MOVAB NOBSQ_FREE(R4),R6      ; Get queue listhead
08 10 198C 4483      BSBB 70$      ; Empty queue
56 10 A4 9E 198E 4484      MOVAB NOBSQ_ATTEN(R4),R6      ; Get queue listhead
02 10 1992 4485      BSBB 70$      ; Empty queue
1D 11 1994 4486      BRB 90$      ; Continue
1996 4487
50 00 B6 0F 1996 4488 70$: REMQUE @R6,R0      ; Get buffer
12 1D 199A 4489      BVS 80$      ; If VS then none
40 A4 42 A5 A0 199C 4490      ADDW UCB$$_DEVBUFSIZ(R5),NOBSW_QUOTA(R4) ; Restore quota
54 DD 19A1 4491      PUSHL R4
00000000 GF 16 19A3 4492      JSB G^COM$$_DRVDEALMEM      ; Deallocate buffer
54 8ED0 19A9 4493      POPL R4
E8 11 19AC 4494      BRB 70$      ; Loop
05 19AE 4495 80$: RSB
19AF 4496
19AF 4497 85$: BUG_CHECK NOBUFPCKT,FATAL
19B3 4498
19B3 4499 ; Complete all associated receive IO packets
19B3 4500
53 18 B4 0F 19B3 4501 90$: REMQUE @NOBSQ_RCVS(R4),R3      ; If VS then queue is empty
09 1D 19B7 4502      BVS 100$
```



```
38 A3 2C 3C 19B9 4503 MOVZWL #SS$ ABORT,IRP$L_MEDIA(R3) ; Set status and size
      FCOB 30 19BD 4504 BSBW IO_DONE ; Complete the request
      F1 11 19C0 4505 BRB 90$ ; Branch to get next IRP
      19C2 4506
      19C2 4507 ; Complete all XMTS
      19C2 4508
      19C2 4509 100$: DSBINT UCBSB DIPL(R5) ; Lock out interrupts
64 A5 03 AA 19C9 4510 BICW #UCBSM_TIM!UCBSM_INT,UCBSW_STS(R5) ;
      19CD 4511 ENBINT ; Reset interrupts
      38 BB 19D0 4512 PUSHR #^M<R3,R4,R5> ; Save registers
56 04 9A 19D2 4513 MOVZBL #DLK$C_USRINT,R6 ; Set up to halt the protocol
57 02 9A 19D5 4514 MOVZBL #DLK$M_STOP,R7
      58 7C 19D8 4515 CLRQ R8
55 0C A4 D0 19DA 4516 MOVL NOB$A_PRO_BUFFER(R4),R5 ; Set protocol buffer
      E61F 30 19DE 4517 BSBW DDCMP
56 28 A5 9E 19E1 4518 MOVAB TF$Q_CMPQ(R5),R6 ; Get queue to complete from
      38 BA 19E5 4519 POPR #^M<R3,R4,R5> ; Restore the registers
51 00 B6 0F 19E7 4520 110$: REMQUE @<R6>,R1 ; Complete all XMT's
      13 1D 19EB 4521 BVS 120$ ; If VS then no more to cmplt
13 0A A1 91 19ED 4522 CMPB XMTQ$B_BUFTYP(R1),S^#DYN$C_BUFIO ; Branch NEQ not a valid buff
      BC 12 19F1 4523 BNEQ 85$
53 0C A1 D0 19F3 4524 MOVL XMTQ$L_IRP(R1),R3 ; Get associated IRP
38 A3 2C 3C 19F7 4525 MOVZWL #SS$ ABORT,IRP$L_MEDIA(R3) ; Set aborted status
      FBBF 30 19FB 4526 BSBW TRANSMIT_IO_DONE ; Complete the I/O
      E7 11 19FE 4527 BRB 110$ ; Branch to get next
50 01 3C 1A00 4528 120$: MOVZWL S^#SS$_NORMAL,R0 ; Assume shutdown
03DC 8F BA 1A03 4529 POPR #^M<R2,R3,R4,R6,R7,R8,R9> ; Save the registers
      05 1A07 4530 RSB
      1A08 4531
```

```
1A08 4533 .SBTTL VALIDATE_P2, Validate P2 buffer parameters
1A08 4534 .SBTTL VALIDATE_P2_TRIB, Validate P2 buffer with Trib param
1A08 4535 .SBTTL VALIDATE_P2_UCB, Validate P2 buffer with UCB
1A08 4536
1A08 4537 :++
1A08 4538 : VALIDATE_P2 - Validate P2 buffer parameters
1A08 4539 : VALIDATE_P2_TRIB - Validate P2 buffer with trib parameters
1A08 4540 : VALIDATE_P2_UCB - Validate P2 buffer with UCB
1A08 4541 :
1A08 4542 : This routine is called to validate the P2 buffer parameters. The parameters
1A08 4543 : are checked against a parameter table which verifies that the minimum value
1A08 4544 : and maximum value is not violated, and that invalid status flags are not set.
1A08 4545 :
1A08 4546 : Inputs:
1A08 4547 :
1A08 4548 : R1 = Address of parameter verification table (VALIDATE_P2 entry only)
1A08 4549 : R2 = Status word from UCB
1A08 4550 : R3 = IRP address
1A08 4551 : R4 = NOB or trib parameter block address for value
1A08 4552 : checking (VALIDATE_P2 entry only)
1A08 4553 : R5 = UCB address
1A08 4554 :
1A08 4555 : IPL = FIPL
1A08 4556 :
1A08 4557 : Outputs:
1A08 4558 :
1A08 4559 : R0 = status return of parameters
1A08 4560 :
1A08 4561 : If no error:
1A08 4562 : R1 = Address of parameter verification table
1A08 4563 : If error:
1A08 4564 : R1 = Bad parameter value
1A08 4565 :
1A08 4566 : All other registers are preserved.
1A08 4567 :
1A08 4568 :--
1A08 4569 .ENABL LSB
1A08 4570 VALIDATE_P2_TRIB:
1A08 4571 PUSH R4
1A08 4572 MOVAB TRIB_PARAM,R1
1A08 4573 MOVL UCB$[NO_BUFFER(R5),R4
1A08 4574 MOVAB NOB$Z_DDCMP(R4),R4
1A08 4575
1A08 4576 BRB 10$
1A08 4577 VALIDATE_P2_UCB:
1A08 4578 PUSH R4
1A08 4579 MOVAB LINE_PARAM,R1
1A08 4580 MOVL UCB$[NO_BUFFER(R5),R4
1A08 4581 10$: BSBB VALIDATE_P2
1A08 4582 POPL R4
1A08 4583 RSB
1A08 4584 .DSABL LSB
1A08 4585
1A08 4586 VALIDATE_P2:
1A08 4587 PUSHR #^M<R1,R2,R3,R5,R6,R7,R8,R9>
1A08 4588
1A08 4589 MOVL IRP$L_SVAPTE(R3),R6

51 54 54 DD 1A08 4571
E6C6 CF 9E 1A0A 4572
00A8 C5 D0 1A0F 4573
54 60 A4 9E 1A14 4574
OC 11 1A18 4575
54 DD 1A1A 4577
E680 CF 9E 1A1C 4578
00A8 C5 D0 1A21 4579
04 10 1A26 4581
54 8ED0 1A28 4582
05 1A2B 4583
1A2C 4584
1A2C 4585
03EE 8F BB 1A2C 4587
56 2C A3 D0 1A30 4588
1A30 4589
```

: Validate P2 buffer with TFB
: Save R4
: Get address of verification table
: Get NOB address
: Get address of block to set
: DDCMP parameters
: Validate P2 buffer with UCB
: Save R4
: Get address of verification table
: Get NOB address
: Do the validation
: Restore R4
: Return to caller
: Validate P2 buffer parameters
: Save registers
: NB:R1 must be on top of stack
: Get system P2 buffer address


```

      03 12 1A34 4590      BNEQ 10$      ; Br if a system buffer
      0085 31 1A36 4591      BRW 150$      ; Else, leave
58 32 A3 3C 1A39 4592 10$: MOVZWL IRPSW_BCNT(R3),R8      ; Get size of P2 buffer
      06 C6 1A3D 4593      DIVL #6,R8      ; Calculate number of parameters
      56 66 D0 1A40 4594      MOVL P2B_L_POINTER(R6),R6      ; Point to start of P2 data
      1A43 4595      ; Loop to check next parameter in P2 buffer
      1A43 4596      ; Loop to check next parameter in P2 buffer
      1A43 4597      ; Loop to check next parameter in P2 buffer
50 86 3C 1A43 4598 30$: MOVZWL (R6)+,R0      ; Get parameter type from P2
      55 86 D0 1A46 4599      MOVL (R6)+,R5      ; Get parameter value from P2
      57 6E D0 1A49 4600      MOVL (SP),R7      ; Get parameter table address
      1A4C 4601      ; Loop to check P2 buffer parameter to Line parameter table
      1A4C 4602      ; Loop to check P2 buffer parameter to Line parameter table
      1A4C 4603      ; Loop to check P2 buffer parameter to Line parameter table
59 87 B0 1A4C 4604 40$: MOVW (R7)+,R9      ; Get parameter + flags
      72 13 1A4F 4605      BEQL 170$      ; Br if end of verify table
50 59 0C 00 ED 1A51 4606      CMPZV #PRM_V_TYPE,#PRM_S_TYPE,R9,R0      ; Parameters match?
      16 13 1A56 4607      BEQL 50$      ; Br if yes
      87 B5 1A58 4608      TSTW (R7)+      ; Skip offset word
      1A5A 4609      SKIP PRM_V_MIN,R9,R7      ; Skip minimum value
      1A60 4610      SKIP PRM_V_MAX,R9,R7      ; Skip maximum value
      1A66 4611      SKIP PRM_V_INVALID,R9,R7      ; Skip invalid flags
      DE 11 1A6C 4612      BRB 40$      ; Try next parameter
      1A6E 4613      ; Match found - nullify if same value & check min,max,valid,invalid
      1A6E 4614      ; Match found - nullify if same value & check min,max,valid,invalid
      1A6E 4615      ; Match found - nullify if same value & check min,max,valid,invalid
51 87 B0 1A6E 4616 50$: MOVW (R7)+,R1      ; Get offset + width
      54 D5 1A71 4617      TSTL R4      ; Is data structure present?
      2B 13 1A73 4618      BEQL 100$      ; Br if not - check values
53 51 02 0E EF 1A75 4619      EXTZV #OFF_V_WIDTH,#OFF_S_WIDTH,R1,R3      ; Get width only
51 51 0E 00 EF 1A7A 4620      EXTZV #OFF_V_VALUE,#OFF_S_VALUE,R1,R1      ; Get offset only
      51 54 C0 1A7F 4621      ADDL R4,R1      ; Calculate address of datum
      1A82 4622      CASE R3,TYPE=B,LIMIT=#1,<-      ; Br to handler
      1A82 4623      60$,-      ; Byte value
      1A82 4624      70$,-      ; Word value
      1A82 4625      80$>      ; Longword value
      1A8C 4626      ; Byte value in structure
      1A8C 4627      ; Byte value in structure
      1A8C 4628      ; Byte value in structure
61 55 91 1A8C 4629 60$: CMPB R5,(R1)      ; Is this the same?
      08 11 1A8F 4630      BRB 90$      ; Check result
      1A91 4631      ; Word value
      1A91 4632      ; Word value
      1A91 4633      ; Word value
61 55 B1 1A91 4634 70$: CMPW R5,(R1)      ; Is this the same?
      03 11 1A94 4635      BRB 90$      ; Check result
      1A96 4636      ; Is this the same?
61 55 D1 1A96 4637 80$: CMPL R5,(R1)      ; Is this the same?
      05 12 1A99 4638 90$: BNEQ 100$      ; Br if no - continue checks
      FA A6 B4 1A9B 4639      CLRW -6(R6)      ; Nullify the parameter code
      1B 11 1A9E 4640      BRB 140$      ; Try next parameter - skip checks
      1AA0 4641      ; Try next parameter - skip checks
05 59 0C E1 1AA0 4642 100$: BBC #PRM_V_MIN,R9,110$      ; Br if no minimum value
      87 55 B1 1AA4 4643      CMPW R5,(R7)+      ; Is the value too small?
      1A 1F 1AA7 4644      BLSSU 170$      ; Br if yes - error
05 59 0D E1 1AA9 4645 110$: BBC #PRM_V_MAX,R9,130$      ; Br if no maximum value
      87 55 B1 1AAD 4646      CMPW R5,(R7)+      ; Is the value too big?
```

05	59	11	1A	1AB0	4647		BGTRU	170\$		; Br if yes - error
	52	0E	E1	1AB2	4648	130\$:	BBC	#PRM_V_INVALID,R9,140\$		; Br if no invalid flags
		87	B3	1AB6	4649		BITW	(R7)?,R2		; Check invalid bits
		08	12	1AB9	4650		BNEQ	170\$		; Br if on - error
	85	58	F5	1ABB	4651	140\$:	SOBGTR	R8,30\$		; Loop if more parameters
				1ABE	4652					
	50	01	3C	1ABE	4653	150\$:	MOVZWL	S^#SS\$_NORMAL,R0		; Set success return
		06	11	1AC1	4654		BRB	180\$		; And return
				1AC3	4655					
	6E	50	3C	1AC3	4656	170\$:	MOVZWL	R0,(SP)		; Return bad parameter code
				1AC6	4657					; * R1 Must be on top of stack
	50	14	3C	1AC6	4658		MOVZWL	#SS\$_BADPARAM,R0		; Set error return
	03EE	8F	BA	1AC9	4659	180\$:	POPR	#^M<R1,R2,R3,R5,R6,R7,R8,R9>		; Restore registers
			05	1ACD	4660		RSB			; Return to caller


```
1ACE 4662 .SBTTL UPDATE_P2, Update UCB/TRIB based on P2 buffer parameters
1ACE 4663
1ACE 4664 :++
1ACE 4665 : UPDATE_P2 - Update UCB/TRIB with P2 buffer parameters
1ACE 4666 :
1ACE 4667 : This routine is called to update the UCB/TRIB with the P2 buffer parameters.
1ACE 4668 : The parameters are stored in the appropriate cells of the UCB/TRIB.
1ACE 4669 :
1ACE 4670 : Inputs:
1ACE 4671 :
1ACE 4672 : R1 = Address of parameter verification table
1ACE 4673 : R3 = IRP address
1ACE 4674 : R4 = NOB or ddcmp parameter block address for storing
1ACE 4675 : R5 = UCB address
1ACE 4676 :
1ACE 4677 : IPL = FIPL
1ACE 4678 :
1ACE 4679 : Outputs:
1ACE 4680 :
1ACE 4681 : R0 = destroyed.
1ACE 4682 : All other registers are preserved.
1ACE 4683 :
1ACE 4684 :--
1ACE 4685
1ACE 4686 UPDATE_P2:
1ACE 4687 PUSHHR #^M<R1,R2,R5,R6,R7,R8,R9> : Update the UCB/DDCMP parametr
1AD2 4688 : Save registers
1AD2 4689 MOVL IRP$L_SVAPTE(R3),R6 : NB:R1 must be on top of stack
1AD6 4690 BEQL 80$ : Get system P2 buffer address
1AD8 4691 MOVZWL IRP$W_BCNT(R3),R8 : Br if no system buffer
1ADC 4692 DIVL #6,R8 : Get size of P2 buffer
1ADF 4693 MOVL P2B_L_POINTER(R6),R6 : Calculate number of parameters
1AE2 4694 : Point to start of data
1AE2 4695 : Loop to get next parameter from P2 buffer
1AE2 4696
1AE2 4697 10$: MOVZWL (R6)+,R0 : Get parameter type from P2
1AE5 4698 MOVL (R6)+,R5 : Get parameter value from P2
1AE8 4699 MOVL (SP),R7 : Get parameter table address
1AEB 4700
1AEB 4701 : Loop to store buffer parameter in UCB/DDCMP parameter block
1AEB 4702
1AEB 4703 20$: MOVW (R7)+,R9 : Get parameter + flags
1AEE 4704 BEQL 70$ : Br if end of verify table
1AF0 4705 EXTZV #PRM_V_TYPE,#PRM_S_TYPE,R9,R1 : Get type field
1AF5 4706 CMPW R0,R7 : Parameters match?
1AF8 4707 BEQL 30$ : Br if yes
1AFA 4708 TSTW (R7)+ : Skip offset word
1AFC 4709 SKIP PRM_V_MIN,R9,R7 : Skip minimum value
1B02 4710 SKIP PRM_V_MAX,R9,R7 : Skip maximum value
1B08 4711 SKIP PRM_V_INVALID,R9,R7 : Skip invalid flags
1B0E 4712 BRB 20$ : Try next parameter
1B10 4713
1B10 4714 : Match found - nullify if same value & check min,max,valid,invalid
1B10 4715
1B10 4716 30$: MOVW (R7)+,R1 : Get offset + width
1B13 4717 EXTZV #OFF_V_WIDTH,#OFF_S_WIDTH,R1,R2 : Get width only
1B18 4718 EXTZV #OFF_V_VALUE,#OFF_S_VALUE,R1,R1 : Get offset only
```

03E6	8F	BB	1ACE	4687	PUSHHR	#^M<R1,R2,R5,R6,R7,R8,R9>	: Update the UCB/DDCMP parametr
			1AD2	4688			: Save registers
56	2C	A3	DO	1AD2	4689	MOVL IRP\$L_SVAPTE(R3),R6	: NB:R1 must be on top of stack
		62	13	1AD6	4690	BEQL 80\$	: Get system P2 buffer address
58	32	A3	3C	1AD8	4691	MOVZWL IRP\$W_BCNT(R3),R8	: Br if no system buffer
58	06	C6	1ADC	4692	DIVL #6,R8		: Get size of P2 buffer
56	66	DO	1ADF	4693	MOVL P2B_L_POINTER(R6),R6		: Calculate number of parameters
			1AE2	4694			: Point to start of data
			1AE2	4695			: Loop to get next parameter from P2 buffer
			1AE2	4696			
50	86	3C	1AE2	4697	10\$: MOVZWL	(R6)+,R0	: Get parameter type from P2
55	86	DO	1AE5	4698	MOVL	(R6)+,R5	: Get parameter value from P2
57	6E	DO	1AE8	4699	MOVL	(SP),R7	: Get parameter table address
			1AEB	4700			
			1AEB	4701			: Loop to store buffer parameter in UCB/DDCMP parameter block
			1AEB	4702			
59	87	B0	1AEB	4703	20\$: MOVW	(R7)+,R9	: Get parameter + flags
	47	13	1AEE	4704	BEQL	70\$	: Br if end of verify table
51	59	0C	1AF0	4705	EXTZV	#PRM_V_TYPE,#PRM_S_TYPE,R9,R1	: Get type field
	51	50	1AF5	4706	CMPW	R0,R7	: Parameters match?
		16	1AF8	4707	BEQL	30\$	: Br if yes
		87	1AFA	4708	TSTW	(R7)+	: Skip offset word
			1AFC	4709	SKIP	PRM_V_MIN,R9,R7	: Skip minimum value
			1B02	4710	SKIP	PRM_V_MAX,R9,R7	: Skip maximum value
			1B08	4711	SKIP	PRM_V_INVALID,R9,R7	: Skip invalid flags
		DB	1B0E	4712	BRB	20\$	: Try next parameter
			1B10	4713			
			1B10	4714			: Match found - nullify if same value & check min,max,valid,invalid
			1B10	4715			
52	51	51	1B10	4716	30\$: MOVW	(R7)+,R1	: Get offset + width
51	51	02	1B13	4717	EXTZV	#OFF_V_WIDTH,#OFF_S_WIDTH,R1,R2	: Get width only
		0E	1B18	4718	EXTZV	#OFF_V_VALUE,#OFF_S_VALUE,R1,R1	: Get offset only

51	54	C0	1B1D	4719		ADDL	R4,R1		; Calculate address of datum
			1B20	4720		CASE	R2,TYPE=B,LIMIT=#1,<-		; Br to handler
			1B20	4721			40\$,-		; Byte value
			1B20	4722			50\$,-		; Word value
			1B20	4723			60\$>		; Longword value
			1B2A	4724					
			1B2A	4725				; Byte, word, longword value in structure	
			1B2A	4726					
61	55	90	1B2A	4727	40\$:	MOVB	R5,(R1)		; Store byte value
	08	11	1B2D	4728		BRB	70\$		; Store word value
61	55	B0	1B2F	4729	50\$:	MOVW	R5,(R1)		
	03	11	1B32	4730		BRB	70\$		
61	55	D0	1B34	4731	60\$:	MOVL	R5,(R1)		; Store longword value
A8	58	F5	1B37	4732	70\$:	SOBGTR	R8,10\$		; Br if more parameters
			1B3A	4733					
03E6	8F	BA	1B3A	4734	80\$:	POPR	#^M<R1,R2,R5,R6,R7,R8,R9>		; Restore registers
		05	1B3E	4735		RSB			; Return to caller


```
1B3F 4737 .SBTTL RETURN_P2, Return UCB/DDCMP buffer parameters
1B3F 4738
1B3F 4739 :++
1B3F 4740 : RETURN_P2 - Return P2 buffer parameters
1B3F 4741 :
1B3F 4742 : This routine is called to return the UCB/DDCMP buffer parameters.
1B3F 4743 :
1B3F 4744 : Inputs:
1B3F 4745 :
1B3F 4746 : R1 = Address of return table (same format as verification table)
1B3F 4747 : R2 = Address of user buffer in which to return the parameters
1B3F 4748 : R3 = IRP address
1B3F 4749 : R4 = NOB or DDCMP paramter block address for storing
1B3F 4750 : R5 = UCB address
1B3F 4751 : IRP$W_BCNT(R3) = Size of transfer
1B3F 4752 :
1B3F 4753 : Outputs:
1B3F 4754 :
1B3F 4755 : R0 = destroyed.
1B3F 4756 : All other registers are preserved.
1B3F 4757 :
1B3F 4758 :--
1B3F 4759
1B3F 4760 RETURN_P2:
1B3F 4761 PUSH  #M<R1,R2,R5,R6,R7,R8> ; Return P2 buffer parameters
1B43 4762 MOV  R2,R6 ; Save registers
1B46 4763 BEQL 60$ ; Get user buffer address
1B48 4764 MOVZWL IRP$W_BCNT(R3),R8 ; Br if no system buffer
1B4C 4765 DIVL #6,R8 ; Get size of buffer
1B4F 4766 ; Calculate the number of param
1B4F 4767 ; Loop to return next parameter
1B4F 4768
1B4F 4769 10$: MOVW (R1)+,R5 ; Get parameter + flags
1B52 4770 BEQL 60$ ; Br if end of verify table
1B54 4771 EXTZV #PRM_V_TYPE,#PRM_S_TYPE,R5,R7 ; Get type field
1B59 4772 MOVW R7,(R6)+ ; Return parameter
1B5C 4773 MOVW (R1)+,R7 ; Get offset + width
1B5F 4774 EXTZV #OFF_V_WIDTH,#OFF_S_WIDTH,R7,R2 ; Get width only
1B64 4775 EXTZV #OFF_V_VALUE,#OFF_S_VALUE,R7,R7 ; Get offset only
1B69 4776 ADDL R4,R7 ; Calculate address of datum
1B6C 4777 CASE R2,TYPE=B,LIMIT=#1,<- ; Br to handler
1B6C 4778 20$,- ; Byte value
1B6C 4779 30$,- ; Word value
1B6C 4780 40$> ; Longword value
1B76 4781
1B76 4782 ; Byte, word, longword value in structure
1B76 4783
1B76 4784 20$: MOVZBL (R7),(R6)+ ; Store byte value
1B79 4785 BRB 50$ ;
1B7B 4786 30$: MOVZWL (R7),(R6)+ ; Store word value
1B7E 4787 BRB 50$ ;
1B80 4788 40$: MOVL (R7),(R6)+ ; Store longword value
1B83 4789 50$: SKIP PRM_V_MIN,R5,R1 ; Skip minimum value
1B89 4790 SKIP PRM_V_MAX,R5,R1 ; Skip maximum value
1B8F 4791 SKIP PRM_V_INVALID,R5,R1 ; Skip invalid flags
1B95 4792 SOBGTR R8,T0$ ; Try for more parameters
1B98 4793
```

01E6	8F	BB	1B3F	4761	PUSH	#M<R1,R2,R5,R6,R7,R8>	; Return P2 buffer parameters
56	52	DO	1B43	4762	MOV	R2,R6	; Save registers
	50	13	1B46	4763	BEQL	60\$	; Get user buffer address
58	32	A3	1B48	4764	MOVZWL	IRP\$W_BCNT(R3),R8	; Br if no system buffer
58	06	C6	1B4C	4765	DIVL	#6,R8	; Get size of buffer
			1B4F	4766			; Calculate the number of param
			1B4F	4767			; Loop to return next parameter
			1B4F	4768			
55	81	B0	1B4F	4769	10\$: MOVW	(R1)+,R5	; Get parameter + flags
	44	13	1B52	4770	BEQL	60\$	; Br if end of verify table
57	55	0C	1B54	4771	EXTZV	#PRM_V_TYPE,#PRM_S_TYPE,R5,R7	; Get type field
	86	57	1B59	4772	MOVW	R7,(R6)+	; Return parameter
	57	81	1B5C	4773	MOVW	(R1)+,R7	; Get offset + width
52	57	02	1B5F	4774	EXTZV	#OFF_V_WIDTH,#OFF_S_WIDTH,R7,R2	; Get width only
57	57	0E	1B64	4775	EXTZV	#OFF_V_VALUE,#OFF_S_VALUE,R7,R7	; Get offset only
	57	54	1B69	4776	ADDL	R4,R7	; Calculate address of datum
		C0	1B6C	4777	CASE	R2,TYPE=B,LIMIT=#1,<-	; Br to handler
			1B6C	4778		20\$,-	; Byte value
			1B6C	4779		30\$,-	; Word value
			1B6C	4780		40\$>	; Longword value
			1B76	4781			
			1B76	4782			; Byte, word, longword value in structure
			1B76	4783			
86	67	9A	1B76	4784	20\$: MOVZBL	(R7),(R6)+	; Store byte value
	08	11	1B79	4785	BRB	50\$	;
86	67	3C	1B7B	4786	30\$: MOVZWL	(R7),(R6)+	; Store word value
	03	11	1B7E	4787	BRB	50\$	;
86	67	DO	1B80	4788	40\$: MOVL	(R7),(R6)+	; Store longword value
			1B83	4789	50\$: SKIP	PRM_V_MIN,R5,R1	; Skip minimum value
			1B89	4790	SKIP	PRM_V_MAX,R5,R1	; Skip maximum value
			1B8F	4791	SKIP	PRM_V_INVALID,R5,R1	; Skip invalid flags
B7	58	F5	1B95	4792	SOBGTR	R8,T0\$	; Try for more parameters
			1B98	4793			

NODRIVER
V04-001

- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 116
RETURN_P2, Return UCB/DDCMP buffer para 6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (79)

01E6 8F BA 1B98 4794 60\$: POPR #^M<R1,R2,R5,R6,R7,R8> ; Restore registers
05 1B9C 4795 RSB ; Return to caller


```
189D 4797 .SBTTL UNPACK_P2_BUF, Unpack a P2 parameter from P2 buffer
189D 4798
189D 4799 :++
189D 4800 : UNPACK_P2_BUF - Unpack a P2 parameter from P2 buffer
189D 4801 :
189D 4802 : Functional description:
189D 4803 :
189D 4804 : This routine is called to get a P2 parameter from the P2 buffer.
189D 4805 :
189D 4806 : Inputs:
189D 4807 :
189D 4808 :     R1 = Parameter type code
189D 4809 :     R3 = IRP address
189D 4810 :     R5 = UCB address
189D 4811 :
189D 4812 : Outputs:
189D 4813 :
189D 4814 :     R0 = Low bit set if specified Parameter type code is found in P2
189D 4815 :     R2 = Parameter value if success else destroyed
189D 4816 :
189D 4817 : All other registers are preserved.
189D 4818 :
189D 4819 :--
189D 4820
189D 4821 UNPACK_P2_BUF:
189D 4822     PUSH    #^M<R5,R6,R7>
189D 4823     MOVL    IRP$L_SVAPTE(R3),R6
189D 4824     BEQL    20$
189D 4825     MOVZWL  IRP$W_BCNT(R3),R7
189D 4826     DIVL    #6,R7
189D 4827     MOVAB   P2B_T_DATA(R6),R6
189D 4828     MOVZWL  S^#SS$ _NORMAL,R0
189D 4829
189D 4830 : Loop to check next parameter in P2 buffer
189D 4831
189D 4832 10$: MOVZWL (R6)+,R5
189D 4833     MOVL    (R6)+,R2
189D 4834     CMPW    R1,R5
189D 4835     BEQL    30$
189D 4836     SOBGTR  R7,10$
189D 4837
189D 4838 20$: CLRL    R0
189D 4839 30$: POPR    #^M<R5,R6,R7>
189D 4840     RSB
189D 4841
189D 4842 XG_END:
189D 4843     .END
```

56 00E0 8F BB 189D 4822 : Unpack P2 buffer
2C A3 D0 18A1 4823 : Save registers
1C 13 18A5 4824 : Get system P2 buffer address
57 32 A3 3C 18A7 4825 : Br if none
57 57 06 C6 18AB 4826 : Get size of P2 buffer
56 0C A6 9E 18AE 4827 : Caculate number of parameters
50 01 3C 18B2 4828 : Point to start of data
18B5 4829 : Assume success

55 86 3C 18B5 4832 : Get parameter type from P2
52 86 D0 18B8 4833 : Get parameter value from P2
55 51 B1 18BB 4834 : Parameters match?
05 13 18BE 4835 : Br if yes
F2 57 F5 18C0 4836 : Br if more parameters

00E0 50 D4 18C3 4838 : Return error
8F BA 18C5 4839 : Restore registers
05 05 18C9 4840 : Return to caller

NODRIVER
Symbol table

M 5

- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 118
6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (80)

\$\$\$	= 00000020	R	02	DEVSM_ODV	= 08000000	
\$\$\$OFF	= 00000009			DEVTIMER	0000164C	R 03
\$\$\$TYP	= 00005479			DLASA_POSTQ	00000004	
\$\$\$WID	= 00004000			DLASA_XMT_INPR	00000000	
\$SOP	= 00000002			DLASC_ADDR_LENGTH	00000008	
ABORTIO	000004DD	R	03	DLASK_ADDR_LENGTH	00000008	
ACBSL_KAST	= 00000018			DLKSB_MAINT	00000008	
ACCESS	00000B0B	R	03	DLKSB_MAXREP	00000002	
ADDCMP_VECTOR	00000078	R	03	DLKSB_MAXSEL	00000003	
ADDFREELIST	00001078	R	03	DLKSB_MRB	00000009	
ALLOC_BUFFER	000007A5	R	03	DLKSB_MSGCNT	00000001	
ALLOC_P2BUF	00000B24	R	03	DLKSB_TRIB	00000000	
ALT_RCVFDT	00000488	R	03	DLKSC_ACTNOTCOM	= 00000009	
ATS_NULL	*****	X	02	DLKSC_CHAR	= 00000005	
BUG\$ NOBUFPCKT	*****	X	03	DLKSC_RCVMSG	= 00000001	
CHECK_BUFS	00000AE5	R	03	DLKSC_REQEBA	= 00000003	
CHECK_P1	00000B11	R	03	DLKSC_START_TIMER	= 00000006	
CHECK_P2	00000AE7	R	03	DLKSC_STOP_TIMER	= 00000007	
CHG_TRIB	00000BE7	R	03	DLKSC_USRINT	= 00000004	
CHG_UCB_NOB	00000B7C	R	03	DLKSC_XMTMSG	= 00000002	
CLASS\$ CLASS_DEF	= 00000028			DLKSM_CLEAR	= 00000001	
CLASS_DDT	= 00000010			DLKSM_DATACRC	= 00000004	
CLASS_DISCONNECT	= 00000018			DLKSM_GLOB	= 00000004	
CLASS_DS_TRAN	= 0000000C			DLKSM_HDRCRC	= 00000002	
CLASS_FORK	= 0000001C			DLKSM_HDRERR	= 00000008	
CLASS_GETNXT	= 00000000			DLKSM_LNTYP	= 00000008	
CLASS_LENGTH	= 00000028			DLKSM_MAINT	= 00000004	
CLASS_POWERFAIL	= 00000020			DLKSM_MNTRCV	= 00000080	
CLASS_PUTNXT	= 00000004			DLKSM_MSGCNT	= 00000010	
CLASS_READERROR	= 00000014			DLKSM_MSGSENT	= 00000001	
CLASS_SETUP_UCB	= 00000008			DLKSM_PRSTERR	= 00000002	
CLEAR_NO_BUFFER	00001811	R	03	DLKSM_QEMPTY	= 00000002	
COM\$DELATTNAST	*****	X	03	DLKSM_QFULERR	= 00000020	
COM\$DRVDEALMEM	*****	X	03	DLKSM_RCVQVR	= 00000020	
COM\$FLUSHATTNS	*****	X	03	DLKSM_REPTIM	= 00000080	
COM\$POST	*****	X	03	DLKSM_REPWAIT	= 00000100	
COM\$SETATTNAST	*****	X	03	DLKSM_SELTIM	= 00000020	
COMP_RCV	00001372	R	03	DLKSM_SELWAIT	= 00000040	
COMP_RCV_ERROR	00001432	R	03	DLKSM_SETDEF	= 00000800	
COMP_RCV_NOBUFFER	00001381	R	03	DLKSM_START	= 00000001	
COM_RCVFDT	000004E5	R	03	DLKSM_STOP	= 00000002	
COM_XMITFDT	000003B3	R	03	DLKSM_STRTRCV	= 00000040	
COP_BUFF	00000409	R	03	DLKSM_TMREXP	= 00008000	
CRBSL_INTD	= 00000024			DLKSM_TRIB	= 00000002	
CXBSC_TRAILER	= 00000004			DLKSM_XMTACK	= 00000200	
CXBSC_HEADER	= 00000048			DLKSV_CRC	= 00000010	
DATA_ERROR	00001426	R	03	DLKSV_DUPLEX	= 00000002	
DC\$ SCOM	*****	X	02	DLKSV_MNTRCV	= 00000007	
DDBSL_DDT	= 0000000C			DLKSV_MSGSENT	= 00000000	
DDCMP	*****	X	03	DLKSV_PRSTERR	= 00000001	
DDCMP\$C_HEADER	= 00000006			DLKSV_RCVACK	= 00000008	
DEF_LINE_PARAM	000000FE	RG	03	DLKSV_SETDEF	= 0000000B	
DEF_LINE_PARAMSZ	= 00000006			DLKSV_STATYP	= 00000000	
DEF_TRIB_PARAM	000000F4	RG	03	DLKSV_TMREXP	= 0000000F	
DEF_TRIB_PARAMSZ	= 0000000A			DLKSV_TRNLK	= 0000000E	
DEVSM_IDV	= 04000000			DLKSV_XMTCMP	= 0000000A	
DEVSM_NET	= 00002000			DLKSV_XMTERR	= 00000004	

[illegible]

Variable	Value	Mode	Address
GF\$C-ERREND	00000029		
GF\$C-ERRSRT	0000001F		
GF\$C-LENGTH	0000002C		
GF\$K-ERREND	00000029		
GF\$K-ERRSRT	0000001F		
GF\$K-LENGTH	0000002C		
GF\$L-BABTMR	0000001B		
GF\$L-SELEND	0000000D		
GF\$L-STRTMR	00000017		
GF\$L-TQE_STS	00000011		
GF\$W-GEB	0000000B		
GF\$W-LSETYP	0000001F		
GF\$W-LSE_BCTRS	00000021		
GF\$W-NEXTCNT	00000005		
GF\$W-RSETYP	00000024		
GF\$W-RSE_BCTRS	00000026		
GF\$W-SELQAI	00000003		
HEADER_ERROR	00001410	R	03
INIT_NO_BUFFER	000007D1	R	03
IOSM-CLR_COUNT	= 00000400		
IOSM-RD_COUNT	= 00000100		
IOSV-ATTNAST	= 00000008		
IOSV-CLR_COUNT	= 0000000A		
IOSV-CTRC	= 00000009		
IOSV-NOW	= 00000006		
IOSV-RD_COUNT	= 00000008		
IOSV-RD_MODEM	= 00000007		
IOSV-SHUTDOWN	= 00000007		
IOSV-STARTUP	= 00000006		
IOS_READLBLK	= 00000021		
IOS_READPBLK	= 0000000C		
IOS_READVBLK	= 00000031		
IOS-SENSEMODE	= 00000027		
IOS-SETCHAR	= 0000001A		
IOS-SETMODE	= 00000023		
IOS-VIRTUAL	= 0000003F		
IOS-WRITELBLK	= 00000020		
IOS-WRITEPBLK	= 0000000B		
IOS-WRITEVBLK	= 00000030		
IOCS\$INITIATE	*****	X	03
IOCS\$MNTVER	*****	X	03
IOCS\$REQCOM	*****	X	03
IOCS\$RETURN	*****	X	03
IO_DONE	000015CB	R	03
IPCS_POWER	= 0000001F		
IPLS-SYNCH	= 00000008		
IPLS-TIMER	= 00000008		
IRPSB_NOFUNC	00000021		
IRPSB_TYPE	= 0000000A		
IRPSL-DIAGBUF	= 0000004C		
IRPSL-MEDIA	= 00000038		
IRPSL-PID	= 0000000C		
IRPSL-SVAPTE	= 0000002C		
IRPSQ-STATION	= 00000040		
IRPSV-DIAGBUF	= 00000007		
IRPSV-FUNC	= 00000001		
IRPSW-BCNT	= 00000032		

NODRIVER
Symbol table

- VAX/VMS DMF32 Async DDMCP Line Driver B 6
16-SEP-1984 00:47:19 VAX/VMS Macro V04-00
6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2

Page 120
(80)

```
IRPSW_BOFF      = 00000030
IRPSW_CHAN      = 00000028
IRPSW_FUNC      = 00000020
IRPSW_QUOTA     = 00000040
IRPSW_STS       = 0000002A
JIBSL_BYTCNT    = 00000020
LINE_PARAM      = 000000A0 R    03
LINE_PRMBUF_SIZ = 00000024
MASKR           = 00000080
MASKL           = 00000000
MAX_RCVS        = 00000004
MFDSB_ADDR      = 00000005
MFDSB_MSGID     = 00000000
MFDSB_NUMB      = 00000004
MFDSB_RESP      = 00000003
MFDSB_LENGTH    = 00000006
MFDSK_LENGTH    = 00000006
MFDSM_QSYNC     = 00004000
MFDSM_SELECT    = 00008000
MFDSW_CNTRLG    = 00000001
MFDSW_TYPLG     = 00000001
NMASC_DPX_FUL    = 00000000
NMASC_DPX_HAL    = 00000001
NMASC_LINCN_LOO = 00000001
NMASC_LINCN_NOR = 00000000
NMASC_LINPR_CON = 00000001
NMASC_LINPR_POI = 00000000
NMASC_LINPR_TRI = 00000002
NMASC_PCCI_MRB  = 00000479
NMASC_PCCI_MST  = 00000AFA
NMASC_PCCI_MTR  = 0000047A
NMASC_PCCI_TRI  = 00000474
NMASC_PCLI_BFN  = 00000451
NMASC_PCLI_BUS  = 00000AF1
NMASC_PCLI_CON  = 00000456
NMASC_PCLI_DUP  = 00000457
NMASC_PCLI_PRO  = 00000458
NMASC_PCLI_RTT  = 00000461
NMASC_STATE_OFF = 00000001
NMASC_STATE_ON  = 00000000
NOSALT_ENTRY    = 00000425 R    03
NOSCANCEL       = 000017E3 R    03
NOSCLASS_DISCONNECT = 00001449 R    03
NOSCLASS_PORTFORK = 000013A0 R    03
NOSCLASS_POWERACTION = 0000144A R    03
NOSCONTROL_INIT = 00000304 R    03
NOSC_CIR_PAR    = FFFFFFF700
NOSC_DATA       = 00000003
NOSC_DCRC       = 00000004
NOSC_DEF_BUF_SIZ = 00000100
NOSC_DEL        = 000000FF
NOSC_DLE        = 00000090
NOSC_DUETIM_TABLE_SIZE = 00000100
NOSC_EMPTY      = 00000000
NOSC_ENQ        = 00000005
NOSC_HCRC       = 00000002
NOSC_HEADER     = 00000001
```

```
NOSC_HEADER_HCRC = 00000008
NOSC_HEADER_LEN  = 00000006
NOSC_LINE_PAR    = FFF3F300
NOSC_NOBUFFER    = 00000004
NOSC_PAD         = 0000FFFF
NOSC_PADS        = 00000005
NOSC_SEAR_MSG    = 00000000
NOSC_SET_LINE    = 00000018
NOSC_SOH         = 00000081
NOSC_SYN         = 00000096
NOSC_XMTERR      = 0000000A
NOSDDT           = 00000000 RG    03
NOSDPT           = 00000000 R    02
NOSFUNCTABLE     = 00000038 R    03
NOSGETNXT        = 000010FA R    03
NOSGL_DPT        = ***** X    03
NOSNULL          = 0000144E R    03
NOSPORT_TRANSITION = 000013E9 R    03
NOSPUTNXT        = 00001274 R    03
NOSRCVFDI        = 000004B5 R    03
NOSREADERROR     = 000013EA R    03
NOSREGDUMP       = 000016DB R    03
NOSSENSEMODEFDT  = 00000849 R    03
NOSSETMODEFDT    = 0000052F R    03
NOSSETUP_UCB     = 000013E8 R    03
NOSSTARTIO       = 00000C20 R    03
NOSUNIT_INIT     = 00000323 R    03
NOSXMITFDT       = 00000344 R    03
NOBSA_PRO_BUFFER = 0000000C
NOBSB_BFN        = 00000073
NOBSB_CON        = 00000072
NOBSB_DUP        = 00000071
NOBSB_PRO        = 00000070
NOBSB_RSTATE     = 00000043
NOBSB_SPD        = 00000076
NOBSB_TYPE       = 0000000A
NOBSB_XSTATE     = 00000042
NOBSC_LENGTH     = 000000AA
NOBSC_SETPRM     = 00000070
NOBSK_LENGTH     = 000000AA
NOBSL_BLINK      = 00000004
NOBSL_FLINK      = 00000000
NOBSL_PID        = 00000058
NOBSL_RCV_DUETIM = 00000038
NOBSL_RCV_INPR   = 00000030
NOBSL_XMT_DUETIM = 0000003C
NOBSL_XMT_INPR   = 00000034
NOBSQ_ATTIN      = 00000010
NOBSQ_FREE       = 00000020
NOBSQ_POST       = 00000028
NOBSQ_RCVS       = 00000018
NOBSW_CHANC      = 0000005E
NOBSW_CHANL      = 0000005C
NOBSW_DEVBUS_SIZ = 00000074
NOBSW_ERROR      = 00000054
NOBSW_FLAGS      = 00000056
NOBSW_INDEX      = 00000052
```


NODRIVER
Symbol table

- VAX/VMS DMF32 Async DDMCP Line Driver C 6 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 121
6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (80)

NOBSW_MSGSIZ	00000050			P2B_T_DATA	0000000C		
NOBSW_PADS	00000044			P2B_W_SIZE	00000008		
NOBSW_QUOTA	00000040			P3	= 00000008		
NOBSW_SIZE	00000008			P4	= 0000000C		
NOBSW_XMTERR_SIZE	00000046			P5	= 00000010		
NOBSZ_CTL_MSG	00000080			PCBSL_JIB	= 00000080		
NOBSZ_DDCMP	00000060			PCBSL_PID	= 00000060		
NOBSZ_DLA_ADDR	00000078			POKE_USER	000016E1	R	03
NOBSZ_HEADER	00000048			PORT_ABORT	= 00000020		
NOBUFFER_ERROR	00001441	R	03	PORT_DISCONNECT	= 00000004		
NO_DS_M_FORK_PEND	= 00000080			PORT_FORKRET	= 00000034		
NO_DS_M_ILOOP_SUP	= 00000100			PORT_RESUME	= 00000024		
NO_DS_M_INITED	= 00000040			PORT_SET_LINE	= 00000008		
NO_DS_M_MSG_SENT	= 00000200			PORT_STARTIO	= 00000000		
NO_DS_M_RCVING	= 00000020			PR\$ IPL	*****	X	03
NO_DS_M_RCV_TIME	= 00001000			PRM_M_INVALID	= 00004000		
NO_DS_M_SHUTDOWN	= 00000400			PRM_M_MAX	= 00002000		
NO_DS_M_XMTING	= 00000010			PRM_M_MIN	= 00001000		
NO_DS_M_XMT_TIME	= 00000800			PRM_M_TYPE	= 00000FFF		
NO_DS_V_FORK_PEND	= 00000007			PRM_S_TYPE	= 0000000C		
NO_DS_V_ILOOP_SUP	= 00000008			PRM_V_INVALID	= 0000000E		
NO_DS_V_INITED	= 00000006			PRM_V_MAX	= 000C000D		
NO_DS_V_MSG_SENT	= 00000009			PRM_V_MIN	= 0000000C		
NO_DS_V_RCVING	= 00000005			PRM_V_TYPE	= 00000000		
NO_DS_V_RCV_TIME	= 0000000C			QUEPKT	00000A70	R	03
NO_DS_V_SHUTDOWN	= 0000000A			RCV_B_BLKTYPE	0000000A		
NO_DS_V_XMTING	= 00000004			RCV_B_FIPL	0000000B		
NO_DS_V_XMT_TIME	= 0000000B			RCV_CXB_SPARE	00000014		
NO_ERROR	0000140B	R	03	RCV_DATA	0000134F	R	03
NO_FC_V_READ_MODEM	= 00000004			RCV_HEADER	000012ED	R	03
NO_FC_V_STOP_CIR	= 00000002			RCV_L_LINK	00000000		
NO_FC_V_STOP_LIN	= 00000003			RCV_M_CNTL	= 00000001		
NO_FC_V_STRT_CIR	= 00000000			RCV_NOBUFFER	00001392	R	03
NO_FC_V_STRT_LIN	= 00000001			RCV_T_DATA	00000048		
NO_FS_M_IOFORK	= 00000002			RCV_V_CNTL	= 00000000		
NO_FS_M_PORTFORK	= 00000001			RCV_W_BLKSIZE	00000008		
NO_FS_M_POWERFAIL	= 00000004			RCV_W_ERROR	00000010		
NO_FS_V_IOFORK	= 00000001			RCV_W_FLAGS	00000012		
NO_FS_V_PORTFORK	= 00000000			RCV_W_INDEX	0000000E		
NO_FS_V_POWERFAIL	= 00000002			RCV_W_MSGSIZ	0000000C		
OFF_M_VALUE	= 00003FFF			RCV_Z_HEADER	00000040		
OFF_S_VALUE	= 0000000E			READ_MODEM	00000D5A	R	03
OFF_S_WIDTH	= 00000002			RECEIVE_DONE	000014DB	R	03
OFF_V_VALUE	= 00000000			RETURN_P2	00001B3F	R	03
OFF_V_WIDTH	= 0000000E			SCH\$GL_PCBVEC	*****	X	03
ORB\$B_FLAGS	= 0000000B			SCHED_FORK_IO	000013B8	R	03
ORB\$B_OWNER	= 00000000			SCHED_FORK_POWERFAIL	000013D0	R	03
ORB\$M_PROT_16	= 00000001			SEARCH_MESSAGE	0000129E	R	03
ORB\$W_PROT	= 00000018			SEND_DATA	00001203	R	03
OVERRON_ERROR	00001420	R	03	SEND_DCRC	00001223	R	03
P1	= 00000000			SEND_HCRC	000011EA	R	03
P2	= 00000004			SEND_HEADER	000011D1	R	03
P2B_B_SPARE	0000000B			SEND_PADS	0000123C	R	03
P2B_B_TYPE	0000000A			SEND_XMTERR	00001255	R	03
P2B_C_LENGTH	0000000C			SENSEMODE_CTRL	00000956	R	03
P2B_L_BUFFER	00000004			SENSE_MODEM	00000A51	R	03
P2B_L_POINTER	00000000			SETMODE_CTRL	00000615	R	03

SET CHAR	0000102B	R	03	TF\$K_ERREND	000001C5
SHUTDOWN_CIRCUIT	000018E0	R	03	TF\$K_ERRSTRT	0000018A
SHUTDOWN_LINE	0000184E	R	03	TF\$K_LENGTH	000001C8
SHUTDOWN_LINE_ALT	00001851	R	03	TF\$SL_DBYTR	0000018C
SIZ...	= 00000002			TF\$SL_DBYTX	00000192
SS\$ ABORT	= 0000002C			TF\$SL_DMSGR	00000198
SS\$ ACCVIO	= 0000000C			TF\$SL_DMSGX	0000019E
SS\$ BADPARAM	= 00000014			TF\$SL_HDR	00000000
SS\$ BUFFEROVF	= 000000601			TF\$SL_QACK	0000005C
SS\$ CTRLERR	= 00000054			TF\$SL_QNAK	0000008C
SS\$ DEACTIVE	= 000002C4			TF\$SL_QREP	0000008C
SS\$ DEVINACT	= 000020D4			TF\$SL_QSTACK	0000011C
SS\$ ENDOFFILE	= 00000870			TF\$SL_QSTRT	000000EC
SS\$ NORMAL	= 00000001			TF\$SL_TQE	0000014C
SS\$ POWERFAIL	= 00000364			TF\$Q_CMPQ	00000028
START_CIRCUIT	00000F86	R	03	TF\$Q_CTLQ	00000020
START_CIRCUIT_COMP	0000101E	R	03	TF\$Q_RTOQ	00000038
START_CIRCUIT_ERROR	00001016	R	03	TF\$Q_XMTOVF	00000040
START_CTRL_ERROR	00000F7D	R	03	TF\$Q_XMTQ	00000030
START_DISP	00000C35	R	03	TF\$W_DBRTYP	0000018A
START_LINE	00000EA6	R	03	TF\$W_DBXTYP	00000190
START_POWERFAIL	00001011	R	03	TF\$W_DEIBC	000001AD
START_RECEIVE	000010CD	R	03	TF\$W_DEITYP	000001AB
START_TRANSMIT	00000C47	R	03	TF\$W_DEOBC	000001A8
TF\$A_BUFPTR	00000048			TF\$W_DEOTYP	000001A6
TF\$A_DEV_TIMER	00000050			TF\$W_DMRTYP	00000196
TF\$A_GFB	0000004C			TF\$W_DMXTYP	0000019C
TF\$A_POSTQ	00000058			TF\$W_LBEB	000001B2
TF\$A_XMT_INPR	00000054			TF\$W_LBETYP	000001B0
TF\$B_A	00000013			TF\$W_LRTOTYP	000001BF
TF\$B_ADDR	0000000D			TF\$W_RBEB	000001B7
TF\$B_CHAR	0000000C			TF\$W_RBETYP	000001B5
TF\$B_CURRTO	0000001B			TF\$W_REPWA	00000018
TF\$B_DEI	000001AF			TF\$W_RRTOTYP	000001C2
TF\$B_DEO	000001AA			TF\$W_SELSP	000001A4
TF\$B_LBE	000001B4			TF\$W_SELTYT	000001A2
TF\$B_LRTO	000001C1			TF\$W_STOBC	000001BC
TF\$B_MAXRTO	0000001A			TF\$W_STOTYP	000001BA
TF\$B_MMCTR	0000001D			TF\$W_TEB	0000001E
TF\$B_MSGCNT	0000001C			TF\$W_THRES	000001C5
TF\$B_N	00000012			TIMEOUT	00001711
TF\$B_NAKRSN	00000188			TIMEOUT_DATA	0000179A
TF\$B_R	00000011			TIMEOUT_DCRC	000017AF
TF\$B_RADDR	0000000F			TIMEOUT_END	000017C1
TF\$B_RBE	000001B9			TIMEOUT_HCRC	0000177B
TF\$B_REPTIM	00000017			TIMEOUT_HEADER	0000175F
TF\$B_RRTO	000001C4			TIMEOUT_PADS	000017B4
TF\$B_SELTIM	00000016			TIMEOUT_XMTERR	000017BD
TF\$B_SFLAGS	00000189			TRANSMIT_DONE	000015EF
TF\$B_STO	000001BE			TRANSMIT_IO_DONE	000015BD
TF\$B_T	00000014			TRIB_PARAM	000000D4
TF\$B_X	00000015			TRIB_PRM_BUFSIZ	= 00000018
TF\$B_XADDR	0000000E			TT\$C_BAUD_110	= 00000003
TF\$B_XQCNT	00000010			TT\$C_BAUD_1200	= 00000008
TF\$C_ERREND	000001C5			TT\$C_BAUD_134	= 00000004
TF\$C_ERRSTRT	0000018A			TT\$C_BAUD_150	= 00000005
TF\$C_LENGTH	000001C8			TT\$C_BAUD_1800	= 00000009

NODRIVER
Symbol table

E 6

- VAX/VMS DMF32 Async DDMCP Line Driver 16-SEP-1984 00:47:19 VAX/VMS Macro V04-00 Page 123
6-SEP-1984 16:22:41 [DRIVER.SRC]NODRIVER.MAR;2 (80)

TTSC_BAUD_19200	= 00000010
TTSC_BAUD_2000	= 0000000A
TTSC_BAUD_2400	= 0000000B
TTSC_BAUD_300	= 00000006
TTSC_BAUD_3600	= 0000000C
TTSC_BAUD_4800	= 0000000D
TTSC_BAUD_50	= 00000001
TTSC_BAUD_600	= 00000007
TTSC_BAUD_7200	= 0000000E
TTSC_BAUD_75	= 00000002
TTSC_BAUD_9600	= 0000000F
TTYSM_PC_DMAENA	= 00000002
TTYSM_PC_NOTIME	= 00000001
TTYSM_PC_PRMMAP	= 00000008
TTYSV_PC_XOFENA	= 00000006
UCBSB_DEVCLASS	= 00000040
UCBSB_DEVTTYPE	= 00000041
UCBSB_DIPL	= 0000005E
UCBSB_FIPL	= 0000000B
UCBSB_TT_OUTTYPE	= 0000010B
UCBSB_TT_PARITY	= 000000F8
UCBSB_TYPE	= 0000000A
UCBSL_TL_LENGTH	= 000000B0
UCBSL_DEVCHAR	= 00000038
UCBSL_DEVDEPEND	= 00000044
UCBSL_DEVDEPN2	= 00000048
UCBSL_FR3	= 00000010
UCBSL_IRP	= 00000058
UCBSL_NO_AST	= 000000AC
UCBSL_NO_BUFFER	= 000000A8
UCBSL_TT_OUTADR	= 0000011C
UCBSL_TT_PORT	= 00000118
UCBSM_INT	= 00000002
UCBSM_ONLINE	= 00000010
UCBSM_POWER	= 00000020
UCBSM_TIM	= 00000001
UCBSQ_TL_BRKTHRU	= 000000A8
UCBSS_TT_DEVSTS_FILL	= 0000000B
UCBSV_INT	= 00000001
UCBSV_ONLINE	= 00000004
UCBSV_POWER	= 00000005
UCBSV_TT_DEVSTS_FILL	= 00000004
UCBSV_TT_HANGUP	= 00000003
UCBSV_TT_NOLOGINS	= 0000000F
UCBSV_TT_NOTIF	= 00000002
UCBSV_TT_TIMO	= 00000001
UCBSW_BOFF	= 0000007C
UCBSW_DEVBUFSIZ	= 00000042
UCBSW_DEVSTS	= 00000068
UCBSW_ERRCNT	= 00000082
UCBSW_REFC	= 0000005C
UCBSW_SIZE	= 00000008
UCBSW_STS	= 00000064
UCBSW_TT_OUTLEN	= 00000120
UCBSW_TT_PRTCTL	= 00000122
UCBSW_TT_SPEED	= 000000F4
UNPACK_P2_BUF	00001B9D R 03

UPDATE_P2	00001ACE R 03
VALIDATE_P2	00001A2C R 03
VALIDATE_P2-TRIB	00001A08 R 03
VALIDATE_P2-UCB	00001A1A R 03
VECSL_INITIAL	= 0000000C
VECSL_UNITINIT	= 00000018
XG_END	00001BCA R 03
XMSM_CHR_CTRL	= 00000040
XMSM_CHR_DMC	= 00000020
XMSM_ERR_START	= 00800000
XMSM_ERR-TRIB	= 00400000
XMSM_STS_ACTIVE	= 00000800
XMSV_CHR_HDPLX	= 00000002
XMSV_CHR_LOOPB	= 00000001
XMSV_CHR_MOP	= 00000000
XMSV_ERR_FATAL	= 00000010
XMSV_ERR_MAINT	= 00000013
XMSV_ERR_START	= 00000017
XMSV_ERR-TRIB	= 00000016
XMSV_STS_ACTIVE	= 0000000B
XMSV_STS-BUFFAIL	= 0000000C
XMSV_STS-DISC	= 0000000E
XMTQSB_BUFTYP	0000000A
XMTQSB_FLAG	0000001F
XMTQSB-MSGHDR	00000024
XMTQSB-SLOT	0000001C
XMTQSC_LENGTH	0000002A
XMTQSK_LENGTH	0000002A
XMTQSL_BACC	00000014
XMTQSL_IRP	0000000C
XMTQSL_TIMEND	00000010
XMTQSM_CONTROL	= 00000004
XMTQSM-INTERNAL	= 00000080
XMTQSM-ONQUEUE	= 00000001
XMTQSM-SELECT	= 00000002
XMTQSQ_LINK	00000000
XMTQSV_CONTROL	= 00000002
XMTQSV-INTERNAL	= 00000007
XMTQSV-SELECT	= 00000001
XMTQSW-BUFLN	00000008
XMTQSW-DCRC	00000022
XMTQSW-ERROR	0000001D
XMTQSW-HCRC	00000020
XMTQSW-MSGOFF	00000018
XMTQSW-MSGSIZE	0000001A

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$AB\$\$	000001C8 (456.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
\$\$\$105_PROLOGUE	00000092 (146.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE
\$\$\$115_DRIVER	00001BCA (7114.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	35	00:00:00.06	00:00:01.42
Command processing	145	00:00:00.55	00:00:04.37
Pass 1	1439	00:00:39.79	00:02:21.41
Symbol table sort	0	00:00:04.88	00:00:18.87
Pass 2	471	00:00:09.88	00:00:35.17
Symbol table output	2	00:00:00.37	00:00:01.56
Psect synopsis output	2	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	2097	00:00:55.55	00:03:22.82

The working set limit was 3150 pages.
316805 bytes (619 pages) of virtual memory were used to buffer the intermediate code.
There were 250 pages of symbol table space allocated to hold 4324 non-local and 353 local symbols.
4843 source lines were read in Pass 1, producing 36 object records in Pass 2.
103 pages of virtual memory were used to define 95 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
_\$255\$DUA28:[DRIVER.OBJ]SYNCHLIB.MLB;1	7
_\$255\$DUA28:[SHRLIB]NMALIBRY.MLB;1	1
_\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	39
_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	11
TOTALS (all libraries)	58

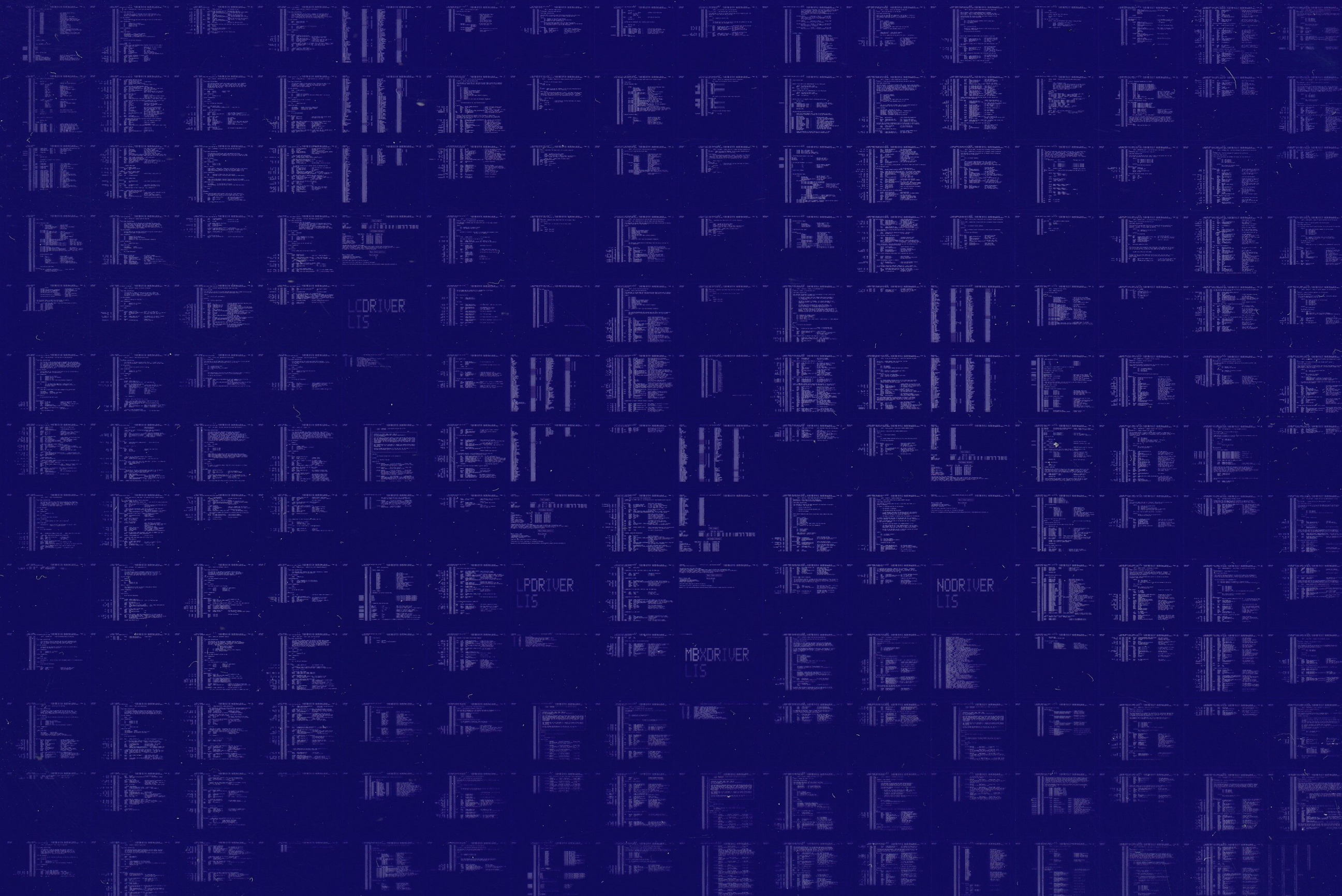
4796 GETS were required to define 58 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:NODRIVER/OBJ=OBJ\$:NODRIVER MSRC\$:NODRIVER/UPDATE=(ENH\$:NODRIVER)+EXECMLS/LIB+SHRLIB\$:NMALIBRY/LIB+LIB\$:SYNCHLIB/LIB

0112 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



0113 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

PA CONFIG
LIS

PA END
LIS

PA ERROR
LIS